Microsoft

Inside Microsoft Dynamics® AX 2012

The Microsoft Dynamics AX Team

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2012 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Control Number: 2012950241
ISBN: 978-0-7356-6710-5

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Microsoft and the trademarks listed at *http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx* are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions Editor: Anne Hamilton
Developmental Editor: Margaret Sherman with the Microsoft Dynamics AX Team
Project Editor: Valerie Woolley
Editorial Production: Christian Holdener, S4Carlisle Publishing Services
Technical Reviewer: Allan Iversen
Copyeditor: Andrew Jones
Indexer: Maureen Johnson, MoJo's Indexing Service
Cover: Twist Creative · Seattle

Contents at a glance

	Foreword	xxiii
	Introduction	xxv
PART I	**A TOUR OF THE DEVELOPMENT ENVIRONMENT**	
CHAPTER 1	Architectural overview	3
CHAPTER 2	The MorphX development environment and tools	19
CHAPTER 3	Microsoft Dynamics AX and .NET	73
CHAPTER 4	The X++ programming language	87
PART II	**DEVELOPING WITH MICROSOFT DYNAMICS AX**	
CHAPTER 5	Designing the user experience	137
CHAPTER 6	The Microsoft Dynamics AX client	159
CHAPTER 7	Enterprise Portal	195
CHAPTER 8	Workflow in Microsoft Dynamics AX	245
CHAPTER 9	Reporting in Microsoft Dynamics AX	275
CHAPTER 10	BI and analytics	299
CHAPTER 11	Security, licensing, and configuration	351
CHAPTER 12	Microsoft Dynamics AX services and integration	385
CHAPTER 13	Performance	417
CHAPTER 14	Extending Microsoft Dynamics AX	493
CHAPTER 15	Testing	527
CHAPTER 16	Customizing and adding help	545
PART III	**UNDER THE HOOD**	
CHAPTER 17	The database layer	577
CHAPTER 18	The Batch framework	613
CHAPTER 19	Application domain frameworks	633

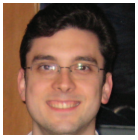
Michael Merz is a program manager for Microsoft Dynamics AX, where he is responsible for the delivery of the Microsoft Dynamics AX services framework and Microsoft Dynamics AX integration capabilities. He has over 15 years of experience in the software industry. Prior to working at Microsoft, Michael held various engineering and management positions in companies including Amazon.com, BEA Systems, and early-stage start-up companies, where he worked on embedded systems, online advertising, social networks, and enterprise software. He has an MSc in computer science from Ulm University, Germany, and lives in Bothell, WA, with his wife, Florina, and his children, Brooke and Joshua.

Amar Nalla is currently a development lead in the Microsoft Dynamics AX product group. He has more than 11 years of experience in the software industry. He started working on the Microsoft Dynamics team during the Axapta 4.0 release. He is part of the foundation team responsible for the Microsoft Dynamics AX server components, and during the past three releases of Microsoft Dynamics AX, he has worked on various components of the server. He maintains a blog at *http://blogs.msdn.com/b/amarnalla/.*

In his spare time, Amar likes to explore the beautiful Puget Sound area.

Parth Pandya is a senior program manager in the Microsoft Dynamics AX product group. For Microsoft Dynamics AX 2012, Parth's area of focus was the new security framework that was built for the release, including the flexible authentication capability and support for Active Directory groups as Microsoft Dynamics AX users. He also contributed to the named user licensing model that was instituted for Microsoft Dynamics AX 2012. Parth has been with Microsoft for over nine years, over five of which were spent working on various releases of the Windows Internet Explorer browser. He particularly enjoyed working as a penetration tester for the number one target of hackers around the world.

Parth swapped the organized chaos of Mumbai, India, for the disorienting tranquility of the Pacific Northwest, where he lives with his wife, Varsha, and three-year-old son, Aarush.



Gustavo Plancarte is a senior software design engineer who joined Microsoft in 2004 after graduating from ITESM in Monterrey, Mexico. He has worked on Microsoft Dynamics AX since version 4.0. On the platform team, he is responsible for driving the common intermediate language (CIL) migration of the X++ programming language, the Software-plus-Services architecture of

Contents

Foreword xxiii

Introduction xxv

The history of Microsoft Dynamics AX xxvi

Who should read this book xxvii

Who should not read this book xxviii

Organization of this book xxviii

Conventions and features in this book xxix

System requirements xxix

Code samples xxx

Acknowledgments xxxi

Errata & book support xxxii

We want to hear from you xxxiii

Stay in touch xxxiii

PART I A TOUR OF THE DEVELOPMENT ENVIRONMENT

Chapter 1 Architectural overview 3

Introduction 3

Microsoft Dynamics AX five-layer solution architecture 4

Microsoft Dynamics AX application platform architecture 6

Application development environments 6

Data tier of the Microsoft Dynamics AX platform 7

Middle tier of the Microsoft Dynamics AX platform 7

Presentation tier of the Microsoft Dynamics AX platform 8

Microsoft Dynamics AX application meta-model architecture 9

Application data element types 10

MorphX user interface control element types 11

Workflow element types 12

Code element types 13
Services element types 13
Role-based security element types 14
Web client element types 14
Documentation and resource element types 16
License and configuration element types 16

Chapter 2 The MorphX development environment and tools 19

Introduction 19
Application Object Tree 20
Navigate through the AOT 21
Create elements in the AOT 23
Modify elements in the AOT 23
Refresh elements in the AOT 25
Element actions in the AOT 25
Element layers and models in the AOT 26
Projects 27
Create a project 27
Automatically generate a project 28
Project types 30
Property sheet 30
X++ code editor 31
Shortcut keys 32
Editor scripts 33
Label editor 33
Create a label 35
Reference labels from X++ 36
Code compiler 37
Best Practices tool 39
Rules 40
Suppress errors and warnings 41
Add custom rules 42

Debugger 43
Enable debugging 43
Debugger user interface 44
Debugger shortcut keys 47
Reverse Engineering tool 47
UML data model 48
UML object model 49
Entity relationship data model 51
Table Browser tool 52
Find tool 53
Compare tool 54
Start the Compare tool 55
Use the Compare tool 57
Compare APIs 58
Cross-Reference tool 60
Version control 62
Element life cycle 64
Common version control tasks 65
Work with labels 66
Synchronize elements 67
View the synchronization log 68
Show the history of an element 69
Compare revisions 70
View pending elements 70
Create a build 71
Integrate Microsoft Dynamics AX with other version control systems 71

Chapter 3 Microsoft Dynamics AX and .NET 73

Introduction 73
Use third-party assemblies 74
Use strong-named assemblies 74
Reference a managed DLL from Microsoft Dynamics AX 75

Code against the assembly in X++ 76
Write managed code 77
Debug managed code 81
Proxies 82
Hot swap assemblies on the server 84

Chapter 4 The X++ programming language 87

Introduction 87
Jobs 88
The type system 88
Value types 88
Reference types 89
Type hierarchies 89
Syntax 93
Variable declarations 93
Expressions 95
Statements 96
Macros 113
Comments 115
XML documentation 116
Classes and interfaces 117
Fields 118
Methods 118
Delegates 120
Pre- and post-event handlers 122
Attributes 123
Code access security 124
Compiling and running X++ as .NET CIL 126
Design and implementation patterns 128
Class-level patterns 129
Table-level patterns 131

PART II DEVELOPING WITH MICROSOFT DYNAMICS AX

Chapter 5 Designing the user experience 137

Introduction 137

A role-tailored design approach 139

User experience components 140

Navigation layer forms 141

Work layer forms 142

Role Center pages 142

Cues 143

Design Role Centers 143

Area pages 144

Design area pages 145

List pages 146

A simple scenario: taking a call from a customer 146

Use a list page as an alternative to a report 148

Design list pages 149

Details forms 150

Transaction details forms 153

Enterprise Portal web client user experience 155

Navigation layer forms 156

Work layer forms 157

Design for Enterprise Portal 157

Design for your users 157

Chapter 6 The Microsoft Dynamics AX client 159

Introduction 159

Working with forms 159

Form patterns 160

Form metadata 162

Form data sources 164
Form queries 170
Adding controls 172
Control overrides 173
Control data binding 173
Design node properties 173
Runtime modifications 174
Action controls 174
Layout controls 176
Input controls 178
ManagedHost control 179
Other controls 181
Using parts 181
Types of parts 181
Reference a part from a form 182
Adding navigation items 182
MenuItem 182
Menu 183
Menu definitions 183
Customizing forms with code 184
Method overrides 184
Auto variables 187
Business logic 188
Custom lookups 188
Integrating with the Microsoft Office client 189
Make data sources available to Office Add-ins 189
Build an Excel template 190
Build a Word template 191
Add templates for users 192

Chapter 7 Enterprise Portal 195

Introduction 195
Enterprise Portal architecture 196

Enterprise Portal components 198
Web parts 199
AOT elements 201
Datasets 201
Enterprise Portal framework controls 203
Developing for Enterprise Portal 216
Create a model-driven list page 217
Create a details page 219
AJAX 222
Session disposal and caching 223
Context 223
Data 225
Metadata 225
Proxy classes 226
ViewState 228
Labels 229
Formatting 230
Validation 231
Error handling 231
Security 232
Secure web elements 233
Record context and encryption 235
SharePoint integration 235
Site navigation 235
Site definitions, page templates, and web parts 237
Import and deploy a web part page 239
Enterprise Search 240
Themes 243

Chapter 8 Workflow in Microsoft Dynamics AX 245

Introduction 245
Microsoft Dynamics AX 2012 workflow infrastructure 246
Windows Workflow Foundation 249

Key workflow concepts 250
Workflow document and workflow document class 250
Workflow categories 251
Workflow types 251
Event handlers 252
Menu items 252
Workflow elements 252
Queues 253
Providers 254
Workflows 255
Workflow instances 256
Work items 256
Workflow architecture 256
Workflow runtime 257
Workflow runtime interaction 258
Logical approval and task workflows 260
Workflow life cycle 262
Implementing workflows 263
Create workflow artifacts, dependent artifacts, and business logic 264
State management 266
Create a workflow category 268
Create the workflow document class 268
Add a workflow display menu item 270
Activate the workflow 270

Chapter 9 Reporting in Microsoft Dynamics AX 275

Introduction 275
Inside the Microsoft Dynamics AX 2012 reporting framework 276
Client-side reporting solutions 276
Server-side reporting solutions 277
Report execution sequence 278
Plan your reporting solution 279

Reporting and users 279
Roles in report development 280
Create production reports 281
Model elements for reports 282
SSRS extensions 285
Microsoft Dynamics AX extensions 286
Create charts for Enterprise Portal 289
Microsoft Dynamics AX chart development tools 289
Integration with Microsoft Dynamics AX 290
Data series 292
Add interactive functions to a chart 294
Override the default chart format 296
Troubleshoot the reporting framework 296
The report server cannot be validated 297
A report cannot be generated 297
A chart cannot be debugged because of SharePoint sandbox issues 297

Chapter 10 BI and analytics 299

Introduction 299
Components of the Microsoft Dynamics AX 2012 BI solution 299
Implementing the prebuilt BI solution 301
Implement the prerequisites 302
Configure an SSAS server 302
Deploy cubes 303
Deploy cubes in an environment with multiple partitions 305
Process cubes 307
Provision users in Microsoft Dynamics AX 308
Customizing the prebuilt BI solution 309
Configure analytic content 310
Customize cubes 311
Extend cubes 319
Creating cubes 323

Identify requirements 324
Define metadata. 325
Generate and deploy the cube 328
Add KPIs and calculations. 333
Displaying analytic content in Role Centers. 333
Provide insights tailored to a persona 334
Choose a presentation tool based on a persona 335
SQL Server Power View reports. 335
Excel 340
Business Overview web part and KPI List web part 341
Develop reports with Report Builder 346
Develop analytic reports by using Visual Studio tools for Microsoft Dynamics AX 346

Chapter 11 Security, licensing, and configuration 351

Introduction 351
Security framework overview 351
Authentication 352
Authorization 353
Data security 356
Develop security artifacts 356
Set permissions for a form 356
Set permissions for server methods 359
Set permissions for controls 359
Create privileges. 359
Assign privileges and duties to security roles 361
Use valid time state tables 362
Validate security artifacts 363
Create users. 363
Assign users to roles 363
Set up segregation of duties rules 364
Create extensible data security policies 364
Data security policy concepts 365

Develop an extensible data security policy 365
Debug extensible data security policies 368
Security coding 369
Table permissions framework 369
Code access security 371
Best practice rules 372
Security debugging 373
Licensing and configuration 376
Configuration hierarchy 378
Configuration keys 378
Use configuration keys 380
Types of CALs 381
Customization and licensing 383

Chapter 12 Microsoft Dynamics AX services and integration 385

Introduction 385
Types of Microsoft Dynamics AX services 387
System services 387
Custom services 388
Document services 392
Security considerations 400
Publish Microsoft Dynamics AX services 400
Consume Microsoft Dynamics AX services 401
Sample WCF client for *CustCustomerService* 402
Consume system services 404
Update business documents 407
Invoke custom services asynchronously 409
The Microsoft Dynamics AX send framework 411
Implementing a trigger for transmission 411
Configure transmission mechanisms 414
Consume external web services from Microsoft Dynamics AX 414
Performance considerations 415

Chapter 13 Performance **417**

Introduction 417

Client/server performance 417

Reduce round-trips between the client and the server. 418

Write tier-aware code 422

Transaction performance. 426

Set-based data manipulation operators 427

Restartable jobs and optimistic concurrency 444

Caching 446

Field lists 456

Field justification 462

Performance configuration options. 462

SQL Administration form 462

Server Configuration form 463

AOS configuration 463

Client configuration 464

Client performance 465

Number sequence caching. 465

Extensive logging. 465

Master scheduling and inventory closing 465

Coding patterns for performance 465

Execute X++ code as CIL. 466

Use parallel execution effectively 466

The SysOperation framework. 467

Patterns for checking to see whether a record exists 472

Run a query only as often as necessary 473

When to prefer two queries over a join 474

Indexing tips and tricks 475

When to use *firstfast* 476

Optimize list pages 476

Aggregate fields to reduce loop iterations 477

Performance monitoring tools 478

Microsoft Dynamics AX Trace Parser 479

Monitor database activity 488
Use the SQL Server connection context to find the SPID or user behind a client session 489
The client access log 490
Visual Studio Profiler 490

Chapter 14 Extending Microsoft Dynamics AX 493

Introduction 493
The SysOperation framework 493
SysOperation framework classes 494
SysOperation framework attributes 495
Comparing the SysOperation and RunBase frameworks 495
RunBase example: *SysOpSampleBasicRunbaseBatch* 496
SysOperation example: *SysOpSampleBasicController* 504
The RunBase framework 510
Inheritance in the RunBase framework 510
Property method pattern 511
Pack-unpack pattern 512
Client/server considerations 516
The extension framework 516
Create an extension 517
Extension example 518
Eventing 520
Delegates 521
Pre and *post* events 522
Event handlers 523
Eventing example 524

Chapter 15 Testing 527

Introduction 527
New unit testing features in Microsoft Dynamics AX 2012 527
Use predefined test attributes 528

Create test attributes and filters 530
Microsoft Visual Studio 2010 test tools 533
Use all aspects of the ALM solution 534
Use an acceptance test driven development approach 535
Use shared steps 536
Record shared steps for fast forwarding 537
Develop test cases in an evolutionary manner 538
Use ordered test suites for long scenarios 539
Putting everything together 540
Execute tests as part of the build process 540
Use the right tests for the job 542

Chapter 16 Customizing and adding help 545

Introduction 545
Help system overview 546
Microsoft Dynamics AX client 547
Help viewer 547
Help server 548
AOS 549
Help content overview 549
Topics 549
Publisher 550
Table of contents 550
Summary page 550
Create content 550
Create a topic in HTML 552
Add labels, fields, and menu items to a topic 559
Make a topic context-sensitive 561
Update content from other publishers 562
Create a table of contents file 563
Create non-HTML content 565
Publish content 567
Add a publisher to the *Web.config* file 569

Publish content to the Help server 570
Set Help document set properties 571
Troubleshoot the Help system 572
The Help viewer cannot display content 572
The Help viewer cannot display the table of contents 573

PART III UNDER THE HOOD

Chapter 17 The database layer 577

Introduction 577
Temporary tables 577
InMemory temporary tables 578
TempDB temporary tables 582
Creating temporary tables 583
Surrogate keys 585
Alternate keys 587
Table relations 588
EDT relations and table relations 588
Foreign key relations 590
The *CreateNavigationPropertyMethods* property 591
Table inheritance 594
Modeling table inheritance 594
Table inheritance storage model 596
Polymorphic behavior 596
Performance considerations 598
Unit of Work 599
Date-effective framework 601
Relational modeling of date-effective entities 601
Support for data retrieval 603
Run-time support for data consistency 604
Full-text support 606

The QueryFilter API 607
Data partitions 610
Partition management 611
Development experience 611
Run-time experience 611

Chapter 18 The batch framework 613

Introduction 613
Batch processing in Microsoft Dynamics AX 2012 613
Common uses of the batch framework 614
Performance 615
Create and execute a batch job 615
Create a batch-executable class 616
Create a batch job 618
Use the Batch API 623
Manage batch execution 625
Configure the batch server 625
Create a batch group 626
Manage batch jobs 628
Debug a batch task 629
Configure AOS for batch debugging 629
Configure Visual Studio for debugging X++ in a batch 630

Chapter 19 Application domain frameworks 633

Introduction 633
The organization model framework 634
How the organization model framework works 634
When to use the organization model framework 636
The product model framework 643
How the product model framework works 643
When to use the product model framework 647

Extending the product model framework 647
The operations resource framework 648
How the operations resource framework works 648
When to use the operations resource framework 652
Extensions to the operations resource framework 652
MorphX model element prefixes for the operations resource framework 654
The dimension framework 654
How the dimension framework works 654
Constrain combinations of values 656
Create values 656
Extend the dimension framework 657
Query data 658
Physical table references 659
The accounting framework 659
How the accounting framework works 660
When to use the accounting framework 662
Extensions to the accounting framework 662
Accounting framework process states 662
MorphX model element prefixes for the accounting framework 663
The source document framework 664
How the source document framework works 664
When to use the source document framework 665
Extensions to the source document framework 666
MorphX model element prefixes for the source document framework 667

Chapter 20 Reflection 669

Introduction 669
Reflection system functions 670
Intrinsic functions 670
typeOf system function 671

classIdGet system function 672
Reflection APIs 673
Table data API 673
Dictionary API 676
Treenodes API 680
TreeNodeType 683

Chapter 21 Application models **687**

Introduction 687
Layers 688
Models 690
Element IDs 692
Create a model 693
Prepare a model for publication 694
Set the model manifest 694
Export the model 695
Sign the model 696
Import model files 697
Upgrade a model 699
Move a model from test to production 700
Create a test environment 701
Prepare the test environment 701
Deploy the model to production 701
Element ID considerations 702
Model store API 703

Appendix: Resources for code upgrade *705*
Index *707*

Foreword

Microsoft Dynamics AX and its predecessor, Axapta, have always benefited from a very active and enthusiastic developer community. Some of those developers are employed by mid-size to large firms that build their business selling solutions built on Microsoft Dynamics AX. Others are in the IT departments of companies using Microsoft Dynamics AX as mission-critical infrastructure.

One of the consistent pieces of feedback I've received from those developers over the years is how the raw power and agility provided by the Microsoft Dynamics AX toolset and metadata environment make them more productive than any other line of business application framework. With Microsoft Dynamics AX 2012, we have taken the productivity and power of that toolset to a whole new level; delivering event-based customization, delta customization of forms, a new editor, date effectivity, and subtype/supertype support, to name just a few.

We continued the journey to expose the power of Microsoft SQL Server Reporting Services (SSRS) and Analysis Services directly within Microsoft Dynamics AX, moving all of the out-of-the-box reports and business intelligence inside the platform.

We back all of that up with almost three times the application footprint of prior versions of Microsoft Dynamics AX, truly making Microsoft Dynamics AX both a powerful developer environment and a rich out-of-the-box suite of applications.

This book focuses on the enhancements in the Microsoft Dynamics AX 2012 toolset and is written by the team that brought you those tools. It's truly an insider's view of the entire Microsoft Dynamics AX development and run-time environment. I hope you enjoy it as much as we enjoyed writing the book and creating the product.

Thanks,

Hal Howard

Head of Product Development, Microsoft Dynamics AX

Corporate Vice President, Microsoft Dynamics Research and Development

Introduction

Microsoft Dynamics AX 2012 represents a new generation of enterprise resource planning (ERP) software. With over 1,000 new features and prebuilt industry capabilities for manufacturing, distribution, services, retail, and public sector, Microsoft Dynamics AX 2012 provides a robust platform for developers to deliver specialized functionality more efficiently to the industries that they support.

Microsoft Dynamics AX 2012 is a truly global solution, able to scale with any business as it grows. It is simple enough to deploy for a single business unit in a single country/region, yet robust enough to support the unique requirements for business systems in 36 countries/regions—all from a single-instance deployment of the software.

For this version of Microsoft Dynamics AX, the entire codebase was analyzed and, where necessary, reengineered, so that the application is built more holistically around a set of unified principles. As Microsoft Technical Fellow Mike Ehrenberg explains:

> The heart of Microsoft Dynamics AX 2012 is a set of unified, natural models that let you see, measure, and change your business. In developing this release, every application concept involved in representing the business in software was reexamined. In each case, limitations that forced workarounds and compromises in older ERP products were lifted, and new capabilities were added to provide an even richer software representation of a business and its structure, processes, and policies. Unified, natural Microsoft Dynamics AX 2012 models make modeling simple businesses fast and easy and yet still provide the richness and flexibility to represent the most complex organizations.

Early adopters have also weighed in on the benefits of Microsoft Dynamics AX 2012:

> Microsoft Dynamics AX 2012 allows us to collaborate within our organization and with our constituents . . . using built-in controls and fund/encumbrance accounting capabilities to ensure compliance with Public Sector requirements . . . and using out-of the-box Business Analytics and Intelligence . . . so executives can make effective decisions in real time.
>
> Mike Bailey
>
> Director of Finance and Information Services
>
> City of Redmond (WA)

With the latest release, developing for and customizing Microsoft Dynamics AX will be easier than ever. Developers will be able to work with X++ directly from within Microsoft Visual Studio and enjoy more sophisticated features in the X++ editor, for example. Also, the release includes more prebuilt interoperability with Microsoft SharePoint Server and SQL Server Reporting Services, so that developers spend less time on mundane work when setting up customer systems.

Guido Van de Velde

Director of MECOMS™

Ferranti Computer Systems

Microsoft Dynamics AX 2012 is substantially different from its predecessor, which can mean a steep learning curve for developers and system implementers who have worked with previous versions. However, by providing a broad overview of the architectural changes, new technologies, and tools for this release, the authors of *Inside Microsoft Dynamics AX 2012* have created a resource that will help reduce the time that it takes for developers to become productive with this version of Microsoft Dynamics AX.

The history of Microsoft Dynamics AX

Historically, Microsoft Dynamics AX encompasses more than 25 years of experience in business application innovation and developer productivity. Microsoft acquired the predecessor of Microsoft Dynamics AX, called Axapta, in 2002, with its purchase of the Danish company Navision A/S. The success of the product has spurred an increasing commitment of research and development resources, which allows Microsoft Dynamics AX to grow and strengthen its offering continuously.

The development team that created Microsoft Dynamics AX 2012 consists of three large teams, two that are based in the United States (Fargo, North Dakota, and Redmond, Washington) and one that is based in Denmark (Copenhagen). The Fargo team focuses on finance and human resources (HR), the Redmond team concentrates on project management and accounting and customer relationship management (CRM), and the Copenhagen team delivers supply chain management (SCM). In addition, a framework team develops infrastructure components, and a worldwide distributed team localizes the Microsoft Dynamics AX features to meet national regulations or local differences in business practices in numerous languages and markets around the world.

To clarify a few aspects of the origins of Microsoft Dynamics AX, the authors contacted people who participated in the early stages of the Microsoft Dynamics AX development cycle. The first question we asked was, "How was the idea of using X++ as the programming language for Microsoft Dynamics AX conceived?"

> We had been working with an upgraded version of XAL for a while called OO XAL back in 1996/1997. At some point in time, we stopped and reviewed our approach and looked at other new languages like Java. After working one long night, I decided that our approach had to change to align with the latest trends in programming languages, and we started with X++.
>
> Erik Damgaard
>
> Cofounder of Damgaard Data

Of course, the developers had several perspectives on this breakthrough event.

> One morning when we came to work, nothing was working. Later in the morning, we realized that we had changed programming languages! But we did not have any tools, so for months we were programming in Notepad without compiler or editor support.
>
> Anonymous developer

Many hypotheses exist regarding the origin of the original product name, Axapta. Axapta was a constructed name, and the only requirement was that the letter X be included, to mark the association with its predecessor, XAL. The X association carries over in the name Microsoft Dynamics AX.

Who should read this book

This book explores the technology and development tools in Microsoft Dynamics AX 2012. It is designed to help new and existing Microsoft Dynamics AX developers by providing holistic and in-depth information about developing for Microsoft Dynamics AX 2012—information that may not be available from other resources, such as SDK documentation, blogs, or forums. It aids developers who are either customizing Microsoft Dynamics AX 2012 for a specific implementation or building modules or applications that blend seamlessly with Microsoft Dynamics AX 2012. System implementers and consultants will also find much of the information useful.

Assumptions

To get full value from this book, you should have knowledge of common object-oriented concepts from languages such as C++, C#, and Java. You should also have knowledge of relational database concepts. Knowledge of Structured Query Language (SQL) and Microsoft .NET technology are also advantageous. Transact-SQL statements are used to perform relational database tasks, such as data updates and data retrieval.

Who should not read this book

This book is not aimed at those who install, upgrade, or deploy Microsoft Dynamics AX 2012. It is also beyond the scope of this book to include details about the sizing of production environments. For more information about these topics, refer to the extensive installation and implementation documentation that is supplied with the product or available on TechNet, MSDN, and other websites.

The book also does not provide instructions for those who configure parameter options within Microsoft Dynamics AX 2012 or the business users who use the application in their day-to-day work. For assistance with these activities, refer to the help that is shipped with the product and available on TechNet at *http://technet.microsoft.com/en-us/library/gg852966.aspx*.

Organization of this book

Although *Inside Microsoft Dynamics AX 2012* does not provide exhaustive coverage of every feature in Microsoft Dynamics AX 2012, it does offer a broad view that will benefit developers as they develop for the product.

This book is divided into three sections, each of which focuses on Microsoft Dynamics AX 2012 from a different angle. Part I, "A tour of the development environment," provides an overview of the Microsoft Dynamics AX 2012 architecture that has been written with developers in mind. The chapters in Part I also provide a tour of the internal Microsoft Dynamics AX 2012 development environment to help new developers familiarize themselves with the designers and tools that they will use to implement their customizations, extensions, and integrations.

Part II, "Developing with Microsoft Dynamics AX 2012," provides the information that developers need in order to customize and extend Microsoft Dynamics AX 2012. In addition to explanations of the features, many chapters include examples, some of which are available as downloadable files that can help you learn how to code for

Microsoft Dynamics AX. For information about how to access these files, see the "Code samples" section, later in this introduction.

Part III, "Under the hood," is largely devoted to illustrating how developers can use the underlying foundation of the Microsoft Dynamics AX 2012 application frameworks to develop their solutions, with a focus on the database layer, system and application frameworks, reflection, and models.

Conventions and features in this book

This book presents information using the following conventions, which are designed to make the information readable and easy to follow.

- Application Object Tree (AOT) paths use backslashes to separate nodes, such as Forms\AccountingDistribution\Methods.
- The names of methods, functions, properties and property values, fields, and nodes appear in italics.
- Registry keys and T-SQL commands appear in capital letters.
- User interface (UI) paths use angle brackets to indicate actions—for example, "On the File menu, point to Tools > Options."
- Boxed elements with labels such as "Note" provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press the Tab key.

System requirements

To work with sample code, you must have the RTM version of Microsoft Dynamics AX 2012 installed. For information about the system requirements for installing Microsoft Dynamics AX 2012, see the Microsoft Dynamics AX 2012 Installation Guide at *http://www.microsoft.com/en-us/download/details.aspx?id=12687*.

You must also have an Internet connection to download the sample files that are provided as supplements to many of the chapters.

Note Some of the features described in this book, such as data partitioning and the EP Chart Control, apply only to the Microsoft Dynamics AX 2012 R2. That is noted where those features are discussed.

Code samples

Most of the chapters in this book include code examples that let you interactively try out the new material presented in the main text. You can download the example code from the following page:

http://go.microsoft.com/FWLink/?Linkid=263524

Follow the instructions to download the InsideDynaAX2012_667105_CompanionContent.zip file.

Installing the code samples

Follow these steps to install the code samples on your computer:

1. Unzip the InsideDynaAX2012_667105_CompanionContent.zip file that you downloaded from the book's website.
2. If prompted, review the displayed end user license agreement. If you accept the terms, select the accept option, and then click Next.

Note If the license agreement doesn't appear, you can access it from the same webpage from which you downloaded the InsideDynaAX2012_667105_CompanionContent.zip file.

Using the code samples

The code examples referenced in each chapter are provided as both .xpo files that you can import into Microsoft Dynamics AX and Visual Studio projects that you can open through the corresponding .csproj files. Many of these examples are incomplete, and you cannot import and run them successfully without following the steps indicated in the associated chapter.

Acknowledgments

We want to thank all the people who assisted us in bringing this book to press. We apologize for anyone whose name we missed.

Microsoft Dynamics product team

Special thanks go to the following colleagues, whom we're fortunate to work with.

Margaret Sherman, who pitched the book to Microsoft Press, provided us with training in how to use templates and style sheets, created a schedule for writing, prodded us when we needed prodding to keep the writing process moving along, and provided editorial feedback on every chapter. Thank you, Margaret. This book absolutely would not have seen the light of day without you!

Mark Baker and Steve Kubis, who contributed ace project management and editing work.

Hal Howard, Richard Barnwell, and Ann Beebe, who sponsored the project and provided resources for it.

We're also grateful to the following members of the product team, who provided us with the reviews and research that helped us refine this book:

Ned Baker
Ian Beck
Andy Blehm
Jim Brotherton
Ed Budrys
Gregory Christiaens
Ahmad El Husseini
Josh Honeyman
Hitesh Jawa
Vijeta Johri
Bo Kampmann
Vinod Kumar
Josh Lange
Mey Meenakshisundaram
Igor Menshutkin
Jatan Modi
Sasha Nazarov
Adrian Orth
Christopher Read (Entirenet)
Bruce Rivard
Gana Sadasivam
Alex Samoylenko
Karen Scipi
Ramesh Shankar
Tao Wang
Lance Wheelwright
Chunke Yang
Arif Kureshy

In addition, we want to thank Joris de Gruyter of Streamline Systems LLC. His SysTestListenerTRX code samples on CodePlex (*http://dynamicsaxbuild.codeplex.com/releases*), with supporting documentation on his blog (*http://daxmusings.blogspot.com/*), and his collaboration as we investigated this approach for executing SysTests from Microsoft Dynamics AX were valuable resources as we prepared the chapter on testing.

Microsoft Press

Another big thank-you goes to the great people at Microsoft Press for their support and expertise throughout the writing and publishing process.

Valerie Woolley, the Content Project Manager for the book, who provided ongoing support and guidance throughout the life of the project.

Anne Hamilton—Acquisitions Editor

Christian Holdener—Production Project Manager with S4Carlisle Publishing Services

Allan Iversen—Technical Reviewer

Andrew Jones—Copy Editor

New arrivals

Finally, we would like to welcome the following youngest members of the Microsoft extended family, the children and grandchildren of the authors who arrived during the months that we were laboring on this book:

Charlie Hendrix Bird
Grace Elizabeth Marie Garty
Gavin Roy Healy
Kayden John Healy
Amrita Nalla
Nilay Pandya

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at oreilly.com:

http://go.microsoft.com/FWLink/?Linkid=263523

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*

PART I

A tour of the development environment

CHAPTER 1 Architectural overview 3

CHAPTER 2 The MorphX development environment and tools 19

CHAPTER 3 Visual Studio tools for Microsoft Dynamics AX . . . 73

CHAPTER 4 The X++ programming language 87

CHAPTER 1

Architectural overview

In this chapter

Introduction 3
The Microsoft Dynamics AX five-layer solution architecture 4
The Microsoft Dynamics AX application platform architecture 6
The Microsoft Dynamics AX application meta-model architecture 9

Introduction

The Microsoft Dynamics AX solution is an enterprise resource planning (ERP) solution that integrates financial resource management, operations resource management, and human resource management processes that can be owned and controlled by multinational, multi-company, and multi-industry organizations, including the public sector. The Microsoft Dynamics AX solution encompasses both the Microsoft Dynamics AX application and the Microsoft Dynamics AX application platform on which it is built. The Microsoft Dynamics AX application platform is designed to be the platform of choice for developing scalable, customizable, and extensible ERP applications in the shortest time possible, and for the lowest cost. The following key architectural design principles make this possible.

- **Separation of concerns** A Microsoft Dynamics AX end-to-end solution is delivered by many development teams working both inside Microsoft, inside the Microsoft partner channel, and inside end-user IT support organizations. The separation of concerns principle realized in the Microsoft Dynamics AX architecture makes this possible by separating the functional concerns of a solution into five globalized, secure layers. This separation reduces functional overlap between the logical components that each team designs and develops.
- **Separation of processes** A Microsoft Dynamics AX end-to-end solution scales to satisfy the processing demands of a large number of concurrent users. The separation of processes principle that is realized in the Microsoft Dynamics AX architecture makes this possible by separating processing into three-tiers—a data tier, a middle tier, and a presentation tier. The Microsoft Dynamics AX Windows client, the Microsoft Dynamics AX Enterprise Portal web client, and the Microsoft Office clients are components of the presentation tier; the Microsoft Dynamics AX Application Object Server (AOS), the Microsoft Dynamics AX Enterprise Portal extensions to Microsoft SharePoint Server, and Microsoft SQL Server Reporting Services (SSRS)

are components of the middle tier; the SQL Server and Microsoft SQL Server Analysis Services (SSAS) are components of the data tier of the Microsoft Dynamics AX platform architecture.

- **Model-driven applications** A Microsoft Dynamics AX application team can satisfy application domain requirements in the shortest time possible. The model-driven application principle that is realized in the Microsoft Dynamics AX architecture makes this possible by separating platform-independent development from platform-dependent development, and by separating organization-independent development from organization-dependent development. With platform-independent development, you can model the structure and specify the behavior of application client forms and reports, of application object entities, and of application data entities that run on multiple platform technologies such as the Microsoft Dynamics AX Windows client, SharePoint Server, SQL Server, and the Microsoft .NET Framework. With organization-independent development, you can use domain-specific reference models, such as the units of measure reference model; domain-specific resource-models, such as the person, product, and location models; and domain-specific workflow models, such as approval and review models, which are relevant to all organizations.

Microsoft Dynamics AX five-layer solution architecture

The Microsoft Dynamics AX five-layer solution architecture, illustrated in Figure 1-1, logically partitions a Microsoft Dynamics AX solution into an application platform layer, a foundation application domain layer, a horizontal application domain layer, an industry application domain layer, and a vertical application domain layer. The components in all architecture layers are designed to meet Microsoft internationalization, localization, and security standards, and all layers are built on the Microsoft technology platform.

Note The layers in the Microsoft Dynamics AX five-layer architecture are different from the model layers that are part of the Microsoft Dynamics AX customization framework described later in this book. Architectural layers are logical partitions of an end-to-end solution. Customization layers are physical partitions of application domain code. For more information, see Chapter 21, "Application models."

The Microsoft Dynamics AX application platform and application domain components are delivered on the Microsoft technology platform. This platform consists of the Windows client, the Office suite of products, Windows Server, SQL Server, SSAS, SSRS, SharePoint Server, the Microsoft ASP.NET web application framework, the .NET Framework, and the Microsoft Visual Studio integrated development environment (IDE).

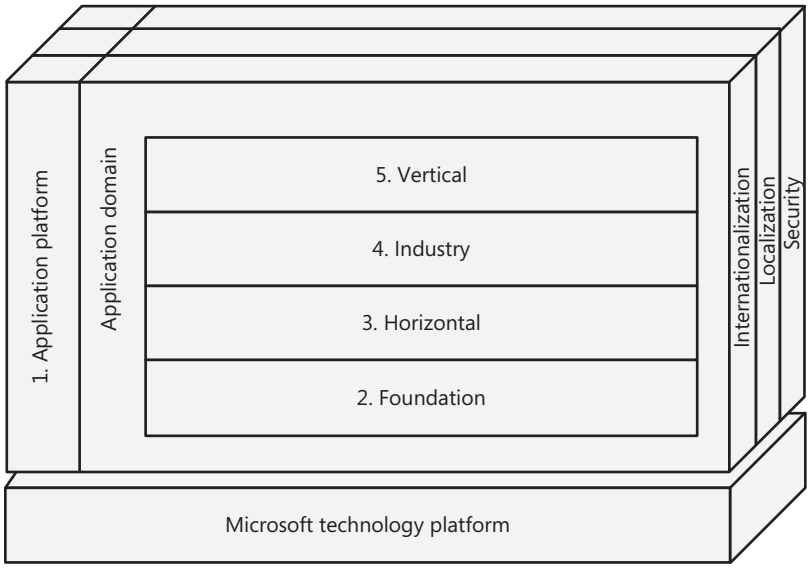


FIGURE 1-1 Microsoft Dynamics AX five-layer architecture.

The following logical partitions are layered on top of the Microsoft technology platform:

- **Layer 1: Application platform layer** The application platform layer provides the system frameworks and tools that support the development of scalable, customizable, and extensible application domain components. This layer consists of the MorphX model-based development environment, the X++ programming language, the Microsoft Dynamics AX Windows client framework, the Enterprise Portal web application framework, the AOS, and the application platform system framework. The architecture of the components in the application platform layer is described in the following section.
- **Layer 2: Foundation application domain layer** The foundation application domain layer consists of domain-specific reference models in addition to domain-specific resource modeling, policy modeling, event documenting, and document processing frameworks that are extended into organization administration and operational domains. Examples of domain-specific reference models include the fiscal calendar, the operations calendar, the language code, and the unit of measure reference models. Examples of domain-specific resource models include the party model, the organization model, the operations resource model, the product model, and the location model. The source document framework and the accounting distribution and journalizing process frameworks are also part of this layer. Chapter 19, "Application frameworks," describes the conceptual design of a number of the frameworks in this layer.

- **Layer 3: Horizontal application domain layer** The horizontal application layer consists of application domain workloads that integrate the financial resource, operations resource, and human resource management processes that can be owned and controlled by organizations. Example workloads include the operations management workload, the supply chain management workload, the supplier relationship management workload, the product information management workload, the financial management workload, the customer relationship management workload, and the human capital management workload. The Microsoft Dynamics AX application can be extended with additional workloads. (The workloads that are part of the Microsoft Dynamics AX solution are beyond the scope of this book.)
- **Layer 4: Industry application domain** The industry application layer consists of application domain workloads that integrate the financial resource, operations resource, and human resource management processes that are specific to organizations that operate in particular industry sectors. Example industries include discrete manufacturing, process manufacturing, distribution, retail, service, and public sector. Workloads in this layer are customized to satisfy industry-specific requirements.
- **Layer 5: Vertical application domain** The vertical application layer consists of application domain workloads that integrate the financial resource, operations resource, and human resource management processes that are specific to organizations that operate in a particular vertical industry and to organizations that are subject to local customs and regulations. Example vertical industries include beer and wine manufacturing, automobile manufacturing, government, and advertising professional services. Workloads in this layer are customized to satisfy vertical industry and localization requirements.

Microsoft Dynamics AX application platform architecture

The architecture of the Microsoft Dynamics AX application platform supports the development of Windows client applications, SharePoint web client applications, Office client integration applications, and third-party integration applications. Figure 1-2 shows the components that support these application configurations. This section provides a brief description of the application development environments, and a description of the components in each of the data, middle, and presentation tiers of the Microsoft Dynamics AX platform architecture.

Application development environments

The Microsoft Dynamics AX application platform includes two model-driven application development environments:

- **Microsoft Dynamics AX MorphX development environment** Use this development environment to develop data models and application code using the Application Object Tree (AOT) application modeling tool and the X++ programming language. This development environment accesses Microsoft Dynamics AX application server services through Microsoft Remote Procedure Call (RPC) technology.

- **Visual Studio** Use this development environment to develop Microsoft .NET plug-ins for and extensions to Microsoft Dynamics AX clients, servers, and services; to develop for Enterprise Portal, and to develop SSRS reports. This development environment accesses the Microsoft Dynamics AX application server services through RPC.

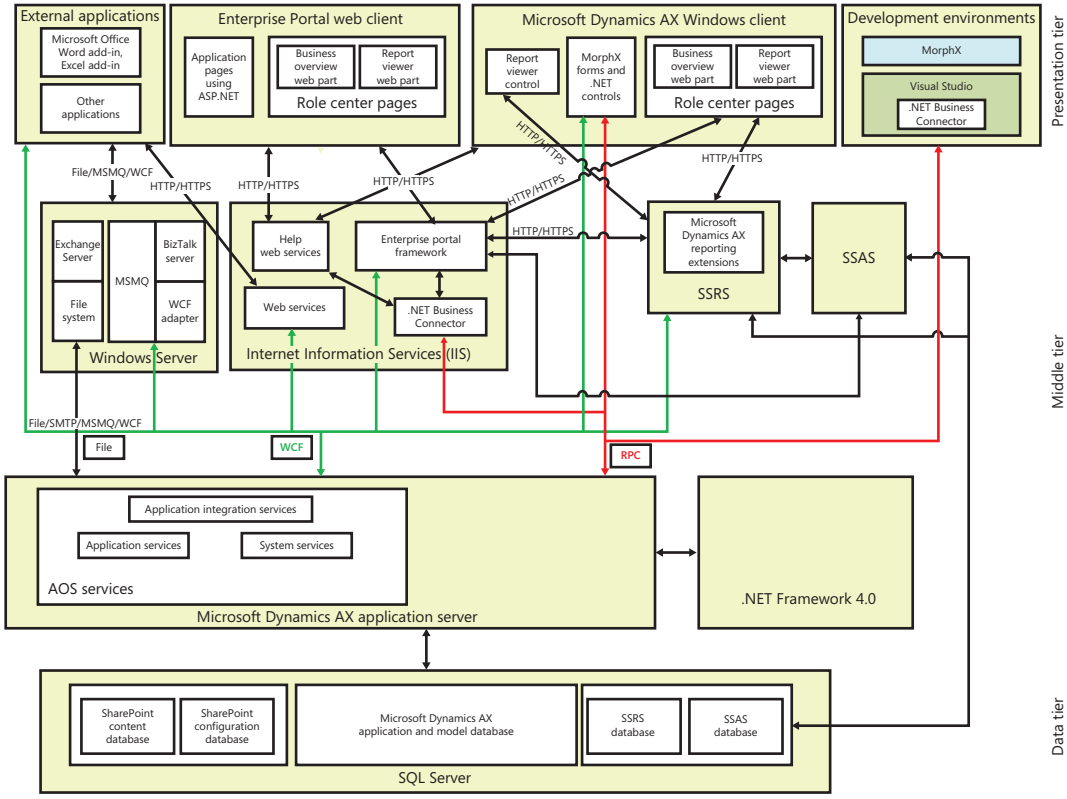


FIGURE 1-2 Architecture of Microsoft Dynamics AX.

Data tier of the Microsoft Dynamics AX platform

The SQL Server database is the only component in the data tier. The database server hosts the SharePoint Server content and configuration databases, the Microsoft Dynamics AX model and application database, the SSRS database, and the SSAS database.

Middle tier of the Microsoft Dynamics AX platform

The middle tier includes the following components:

- **AOS** The AOS executes MorphX application services that are invoked through RPC technology and Windows Communication Foundation (WCF) technology in the .NET Framework. The AOS can be hosted on one computer, but it can also scale out to many computers when additional concurrent user sessions or dedicated batch servers are required.

- **.NET Framework components** These components can be referenced in the AOT so that their application programming interfaces are accessed from X++ programs. The Windows Workflow Foundation (WWF) component is integral to the Microsoft Dynamics AX workflow framework, and WCF is integral to the Microsoft Dynamics AX application integration framework.
- **SQL Server Analysis Services (SSAS)** These services process requests for analytics data hosted by the SQL Server component in the data tier.
- **SSRS and Microsoft Dynamics AX reporting extensions** The reporting extensions provide SSRS with features that are specific to the Microsoft Dynamics AX application platform. These extensions access the AOS through WCF services and access SSAS through HTTP and HTTPS.
- **Microsoft Dynamics AX Enterprise Portal framework** This framework extends the SharePoint application platform with features that are specific to the Microsoft Dynamics AX application platform. The Enterprise Portal framework composes SharePoint content with Microsoft Dynamics AX content accessed from the AOS through the .NET Business Connector and RPC, and content accessed from SSAS and SSRS through HTTP and HTTPS. Enterprise Portal is typically hosted on its own server or in a cluster of servers.
- **Microsoft Dynamics AX Help web service** This web service processes requests for Help content.
- **Web services hosted by Microsoft Internet Information Services (IIS)** The Microsoft Dynamics AX system services can be deployed to and hosted by IIS.
- **Application Integration services** These services provide durable message queuing and transformation services for integration clients.

Presentation tier of the Microsoft Dynamics AX platform

The presentation tier consists of the following components:

- **Windows client** This client executes Microsoft Dynamics AX MorphX and .NET programs developed in MorphX and Visual Studio. The client application communicates with the AOS primarily by using RPC. The client composes navigation, action pane, area page, and form controls for rapid data entry and data retrieval. Form controls have built-in data filtering and search capabilities and their content controls are arranged automatically by the Intellimorph rendering technology. The client additionally hosts role center pages rendered in a web browser control.
- **Enterprise Portal web client** This client executes MorphX application models, X++ programs, and .NET Framework programs developed in the MorphX development environment, Visual Studio, and the SharePoint Server framework. Enterprise Portal is hosted by the Microsoft Dynamics AX runtime, the ASP.NET runtime, and the SharePoint runtime environments. SharePoint and ASP.NET components communicate by means of the Microsoft Dynamics AX .NET Business Connector.

- **Office clients** The Microsoft Word client and Microsoft Excel client are extended by add-ins that work with the Microsoft Dynamics AX platform.
- **Third-party clients** These clients integrate with the Microsoft Dynamics AX platform by means of integration service components such as the file system, Microsoft Message Queuing (MSMQ), Microsoft BizTalk Server, and a WCF adaptor.

Microsoft Dynamics AX application meta-model architecture

Microsoft Dynamics AX application meta-model architecture is based on the principle of model-driven application development. You declaratively program an application by building a model of application components instead of procedurally specifying their structure and behavior with code. The Microsoft Dynamics AX development environment supports both model-driven and code-driven application development.

A model of an application model is called a *meta-model*. Figure 1-3 shows the element types in the Microsoft Dynamics AX application meta-model that you use to develop Microsoft Dynamics AX Windows client applications.

Note To keep the diagram simple, the figure does not list all type dependencies on model element types.

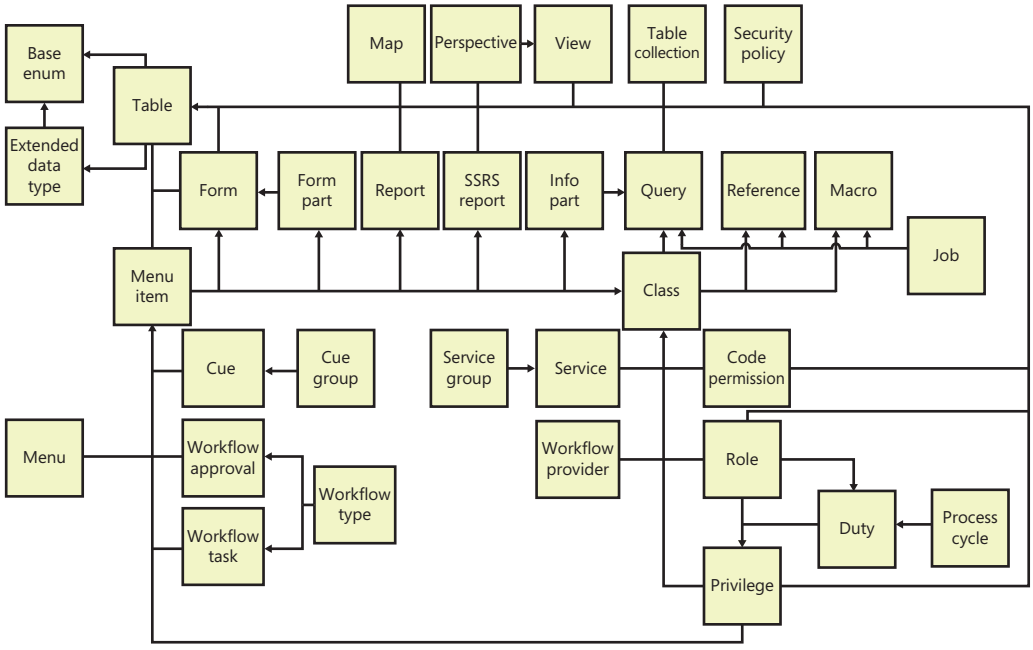


FIGURE 1-3 Element types of the Microsoft Dynamics AX meta-model for developing Microsoft Dynamics AX Windows client applications.

Application data element types

The following model element types are part of the Microsoft Dynamics AX application data meta-model:

- **Base enum** Use a base enumeration (base enum) element type to specify value type application model elements whose fields consist of a fixed set of symbolic constants. For example, you can create a base enum named *WeekDay* to name a set of symbolic constants that includes Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday.
- **Extended data type** Use an extended data type element type to specify value type application model elements that extend base enums, in addition to *string, boolean, integer, real, date, time, UtcDateTime, int64, guid,* and *container* value types. The Microsoft Dynamics AX runtime uses the properties of an extended data type to generate a database schema and to render user interface controls. For example, you could specify an *account number* extended data type as an extension to a *string* value type that is limited to 10 characters in length, and that is described using the *Account number* label when bound to a user interface text entry control. Extended data types also support inheritance. For example, an extended data type that defines an account number can be specialized by other extended data types to define customer and vendor account numbers. The specialized extended data type inherits properties, such as string length, label text, and help text. You can override some of the properties on the specialized extended data type.
- **Table** Use a table element type to specify data entity types that the Microsoft Dynamics AX application platform uses to generate a SQL Server database table schema. Tables specify data entity type fields along with their base enum or extended data type, field groups, indexes, relationships, delete actions, and methods. Tables can also inherit the fields of base tables that they are specified to extend. The Microsoft Dynamics AX runtime uses table specifications to render data entry presentation controls and to maintain the referential integrity of the data stored in the application database. The X++ editor also uses table elements to provide IntelliSense information when you write X++ code that manipulates data stored in the application database. Tables can be bound to form, report, query, and view data sources.
- **Map** Use a map element type to specify a data entity type that factors out common table fields and methods for accessing data stored in horizontally partitioned tables. For example, the *CustTable* and *VendTable* tables in the Microsoft Dynamics AX application model are mapped to the *DirPartyMap* map element so that you can use one *DirPartyMap* object to access common address fields and methods.

Note Consider table inheritance as an alternative to using maps because it increases the referential integrity of a database when base tables are referenced in table relationships.

- **View** Use a view element type to specify a database query that the Microsoft Dynamics AX application platform uses to generate a SQL Server database view schema. Views can include

a query model element that filters data accessed from one table or from multiple joined tables. Views also include table field mappings and methods. Views are read-only and primarily provide an efficient method for reading data. Views can be bound to form, report, and query data sources.

- **Perspective** Use a perspective element type to specify a group of tables and views that are used together when designing and generating SSAS unified dimensional models.
- **Table collection** Use a table collection element type to specify a group of tables whose data is shared by two or more Microsoft Dynamics AX companies assigned to the same virtual company. An application administrator maintains virtual companies, their effective company assignments, and their table collection assignments. The Microsoft Dynamics AX runtime uses the virtual company data area identifier instead of the effective company data area identifier to securely access data stored in tables grouped by a table collection.

Caution The tables in a table collection should only reference tables inside the table collection unless you write application extensions to maintain the referential integrity of the database.

- **Query** Use a query element type to specify a database query. You add tables to query element data sources and specify how they should be joined. You also specify how data is returned from the query by using sort order and range specifications.

MorphX user interface control element types

The following model element types are part of the Microsoft Dynamics AX MorphX user interface control meta-model:

- **Menu item** Use a menu item element type to specify presentation control actions that change the state of the Microsoft Dynamics AX system or user interface or that generate reports. If you specify a label for the menu item, the Microsoft Dynamics AX runtime uses it to name the action when it is rendered in the user interface. The Microsoft Dynamics AX form engine also automatically adds a View details menu item to a drop-down menu, a menu that appears when a user right-clicks a cell in a column that is bound to a table field that is specified as a foreign key in a table relationship. The Microsoft Dynamics AX runtime uses the referenced table's menu item binding to open the form that renders the data from the table. The Microsoft Dynamics AX form and report rendering engines ignore menu items that are disabled by configuration keys or role-based access controls.
- **Menu** Use a menu element type to specify a logical grouping of menu items. Menu specifications can also group submenus. The menu element named *MainMenu* specifies the menu grouping for the Microsoft Dynamics AX navigation pane.

- **Form** Use a form element type to specify a presentation control that a user uses to insert, update, and read data stored in the application database. A form binds table, view, and query data sources to presentation controls. A form is opened when a user selects a control bound to a menu item, such as a button.
- **Form part** Use a form part element type to specify a presentation control that renders a form in the FactBox area of the user interface. For more information about the FactBox area, see Chapter 6, "Designing the user experience."
- **Info part** Use an info part element type to specify a presentation control that renders the result set of a query in the FactBox area of the user interface.
- **Report** Use a report element type to specify a presentation control that renders database data and calculated data in a page-layout format. A user can send a report to the screen, a printer, a printer archive, an email account, or the file system. A report specification binds data sources to presentation controls. A report is opened when a user clicks an output menu item control, such as a button.
- **SSRS report** Use an SSRS report element type to reference a Visual Studio Report Project that is added to the Microsoft Dynamics AX model database.
- **Cue** Use a cue element type to bind a menu item to a presentation control that renders a pictorial representation of a numeric metric, such as the number of open sales orders. A cue is rendered in a Microsoft Dynamics AX Role Center webpage.
- **Cue group** Use a cue group element type to specify a group of cues that are displayed together on the Microsoft Dynamics AX Role Center web part.

Workflow element types

Workflow element types define the workflow tasks, such as review and approval, by binding the tasks to menu items. When a form is workflow-enabled, it automatically renders controls that support the user in performing the tasks in the workflow. Workflow elements define workflow documents and event handlers by using class elements. The following model element types are part of the Microsoft Dynamics AX workflow meta-model:

- **Workflow type** Use a workflow type element type to specify a workflow for processing workflow documents. A workflow configuration consists of event handler specifications, custom workflow task specifications, and menu item bindings.
- **Workflow task** Use a workflow task element type to specify a workflow task. A workflow task comprises a list of task outcomes, event handler registrations, and menu item bindings.
- **Workflow approval** Use a workflow approval element type to specify specialized workflow approval tasks. A workflow approval task consists of approve, reject, request change, and deny task outcomes, a list of event handler registrations, and menu item bindings.

- **Workflow provider** Use a workflow provider element type to specify the name of a class that provides data to a workflow. Example data includes a list of workflow participants, a list of task completion dates, and a structure of users that reflect positions in a position-reporting hierarchy.

Code element types

The following model element types are part of the Microsoft Dynamics AX code meta-model:

- **Class** Use a class element type to specify the structure and behavior of custom X++ types that implement data maintenance, data tracking, and data processing logic in a Microsoft Dynamics AX application. You specify class declarations, methods, and event handlers by using the X++ programming language. Class methods can be bound to menu items so that they are executed when users select action, display, or output menu item controls on a user interface. You can also use a class model element type to specify class interfaces that only include method definitions.
- **Macro** Use a macro element type to specify a library of X++ syntax replacement procedures that map X++ input character sequences, such as readable names, to output character sequences, such as numeric constants, during compilation.
- **Reference** Use a reference element type to specify the name of a. NET Framework assembly that contains .NET Framework common language runtime (CLR) types that can be referenced in X++ source code. The MorphX editor reads type data from the referenced assemblies so that IntelliSense is available for CLR namespaces, types, and type members. The MorphX compiler uses the CLR type definitions in the referenced assembly for type and member syntax validation, and the Microsoft Dynamics AX runtime uses the reference elements to locate and load the referenced assembly.
- **Job** Use a job element type to specify an X++ program that runs when you select the Command\Go menu item or press F5. Developers often write jobs when experimenting with X++ language features. You should not use jobs to write application code.

Services element types

The following model element types are part of the Microsoft Dynamics AX services meta-model:

- **Service** Use a service element type to enable an X++ class to be made available on an integration port.
- **Service group** Use a service group element type to specify a web service deployment configuration that exposes web service operations as basic ports with web addresses.

Role-based security element types

The following model element types are part of the Microsoft Dynamics AX role-based access control security meta-model:

- **Security policy** Use a security policy element type to specify a configuration for constraining the view that a user has of data stored in one or more tables. A security policy configuration consists of a primary table specification and a policy query.
- **Code permission** Use a code permission element type to specify one or more access permissions that secure access to logical units of application data and functionality. You can specify data access permissions to secure access to data stored in tables. You can specify code access permissions to secure access to forms, web controls, and server methods.
- **Privilege** Use a privilege element type to specify one or more permissions that a user requires to perform a task, such as a data maintenance task; or a step in a task, such as a data view or data deletion step.
- **Duty** Use a duty element type to specify a set of privileges that are required for a user to carry out his or her internal control approval, review, and inquiry responsibilities and data maintenance responsibilities.
- **Role** Use a role element type to specify the organization role, functional role, or application role that a user is assigned to in an organization. Sales agent is an example of an organization role, manager is an example of a functional role, and system user role is an example of an application role.
- **Process cycle** Use a process cycle element type to specify the operations and administration activities that are repetitively performed by users who are assigned duties in the security model. The expenditure cycle, the revenue cycle, the conversion cycle, and the accounting cycle are examples of process cycles.

Web client element types

The elements of the Microsoft Dynamics AX application meta-model that are used to develop Enterprise Portal web client applications are illustrated in Figure 1-4.

The following model element types are part of the Microsoft Dynamics AX web client meta-model:

- **Web menu item** Use a menu item element type to specify web navigation actions that change the state of the Microsoft Dynamics AX system or user interface. If a label is specified for the menu item, the Microsoft Dynamics AX runtime will use it to name the action when it is rendered in the user interface.
- **Web menu** Use a web menu element type to specify a logical grouping of web menu items. Web menu specifications can group submenus. Web menus are rendered as hyperlinks on webpages.

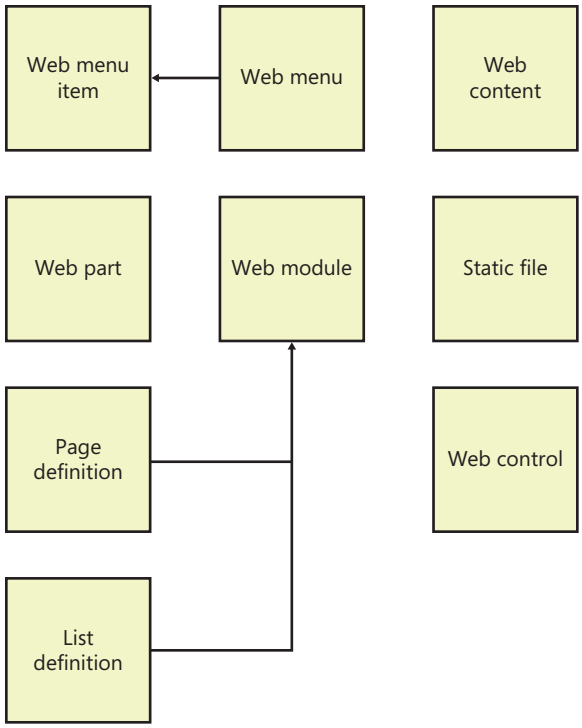


FIGURE 1-4 Element types of the Microsoft Dynamics AX meta-model for developing Enterprise Portal web applications.

- **Web content** Use a web content element type to reference an ASP.NET user control. ASP.NET user controls are developed in the Visual Studio IDE and are stored in the Microsoft Dynamics AX model database.
- **Web part** Use a web part element type to store a SharePoint web part in the Microsoft Dynamics AX model database. The web part will be saved to a web server when deployed.
- **Page definition** Use a page definition element type to store a SharePoint webpage in the Microsoft Dynamics AX model database. The page definition will be saved to a web server when deployed.
- **Web control** Use a web control element type to store an ASP.NET user control in the Microsoft Dynamics AX model database. The web controls will be saved to a web server when deployed.
- **List definition** Use a list definition element type to store a SharePoint list definition in the Microsoft Dynamics AX model database. The list definition will be created on a SharePoint server when deployed.
- **Static file** Use a static file element type to store a file in the Microsoft Dynamics AX model database. The file will be saved to a SharePoint server when deployed.
- **Web module** Use a web module element type to specify the structure of a SharePoint website. The web modules are created as subsites under the home site in SharePoint.

Documentation and resource element types

Documentation and resource element types are used to reference help documentation and system documentation and to develop localized string resources and information resources.

The following model element types are part of the Microsoft Dynamics AX documentation and resource meta-model:

- **Help document set** Use a help documentation set element type to reference a collection of published documents. Help document sets are opened from the Help menu of the Microsoft Dynamics AX Windows client. For more information about creating and updating help documents, see Chapter 16, "Customizing and adding help."
- **System documentation** Use a system documentation element type to reference system library content and hyperlinks to MSDN content. System content describes the Microsoft Dynamics AX system reserved words, functions, tables, types, enums, and classes.
- **Label file** Use a label file element type to store files of localized text resources in the Microsoft Dynamics AX model store.
- **Resource** Use a resource element type to store file resources such as image files and animation files. These resources are stored in the Microsoft Dynamics AX model database.

License and configuration element types

The element types of the Microsoft Dynamics AX application meta-model that are used to develop license, configuration, and application model security are illustrated in Figure 1-5. These model element types change the operational characteristics of the Microsoft Dynamics AX development and runtime environments.

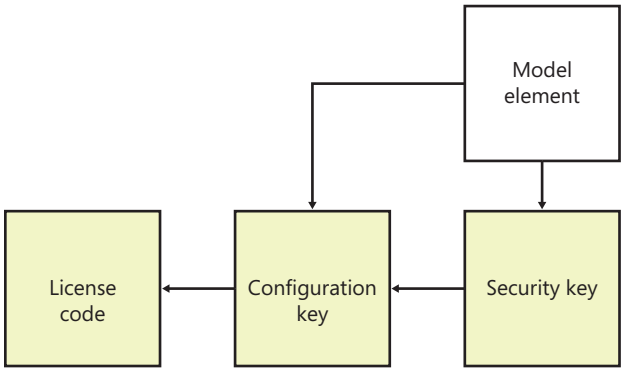


FIGURE 1-5 Element types of the Microsoft Dynamics AX meta-model for developing licensed and configurable application modules.

The following model element types are part of the Microsoft Dynamics AX license, configuration, and application model security meta-model:

- **Configuration key** Use a configuration key element type to assign application model elements to modules that a system administrator then uses to enable and disable application modules and module features. The Microsoft Dynamics AX runtime renders presentation controls that are bound to menu items with active configuration keys. Configuration keys can be specified as subkeys of parent keys.
- **License code** Use a license code element type to lock or unlock the configuration of application modules developed by Microsoft. Modules are locked with license codes that must be unlocked with license keys. License codes can be specified as subcodes of parent codes.

CHAPTER 2

The MorphX development environment and tools

In this chapter

Introduction 19
Application Object Tree 20
Projects 27
Property sheet 30
X++ code editor 31
Label editor 33
Code compiler 37
Best Practices tool 39
Debugger 43
Reverse Engineering tool 47
Table Browser tool 52
Find tool 53
Compare tool 54
Cross-Reference tool 60
Version control 62

Introduction

Microsoft Dynamics AX includes a set of tools, the MorphX development tools, that you can use to build and modify Microsoft Dynamics AX business applications. Each feature of a business application uses the application model elements described in Chapter 1, "Architectural overview." With the MorphX tools, you can create, view, modify, and delete the application model elements, which contain metadata, structure (ordering and hierarchies of elements), properties (key and value pairs), and X++ code. For example, a table element includes the name of the table, the properties set for the table, the fields, the indexes, the relations, the methods, and so on.

This chapter describes the most commonly used tools and offers some tips and tricks for working with them. You can find additional information and an overview of other MorphX tools in the MorphX Development Tools section of the Microsoft Dynamics AX software development kit (SDK) 2012 on MSDN.

Tip To enable development mode in Microsoft Dynamics AX 2012, press Ctrl+Shift+W to launch the Development Workspace, which holds all of the development tools.

Table 2-1 lists the MorphX tools that are discussed in this chapter.

TABLE 2-1 MorphX tools and other components used for development.

Tool	Use this tool to
Application Object Tree (AOT)	Start development activities. The AOT is the main entry point for most development activities. It allows for browsing the repository of all elements that together make up the business application. You can use the AOT to invoke the other tools and to inspect and create elements.
Projects	Group related elements into projects.
Property sheet	Inspect and modify properties of elements. The property sheet shows key and value pairs.
X++ code editor	Inspect and write X++ source code.
Label editor	Create and inspect localizable strings.
Compiler	Compile X++ code into an executable format.
Best Practices tool	Automatically detect defects in both your code and your elements.
Debugger	Find bugs in your X++ code.
Reverse Engineering tool	Generate Microsoft Visio Unified Modeling Language (UML) and Entity Relationship Diagrams (ERDs) from elements.
Table Browser tool	View the contents of a table directly from a table element.
Type Hierarchy Browser and Type Hierarchy Context	Navigate and understand the type hierarchy of the currently active element.
Find tool	Search for code or metadata patterns in the AOT.
Compare tool	See a line-by-line comparison of two versions of the same element.
Cross-Reference tool	Determine where an element is used.
Version control	Track all changes to elements and see a full revision log.

You can access these development tools from the following places:

- In the Development Workspace, on the Tools menu.
- On the context menu of elements in the AOT.

You can personalize the behavior of many MorphX tools by clicking Options on the Tools menu. Figure 2-1 shows the Options form.

Application Object Tree

The AOT is the main entry point to MorphX and the repository explorer for all metadata. You can open the AOT by clicking the AOT icon on the toolbar or by pressing Ctrl+D. The AOT icon looks like this:

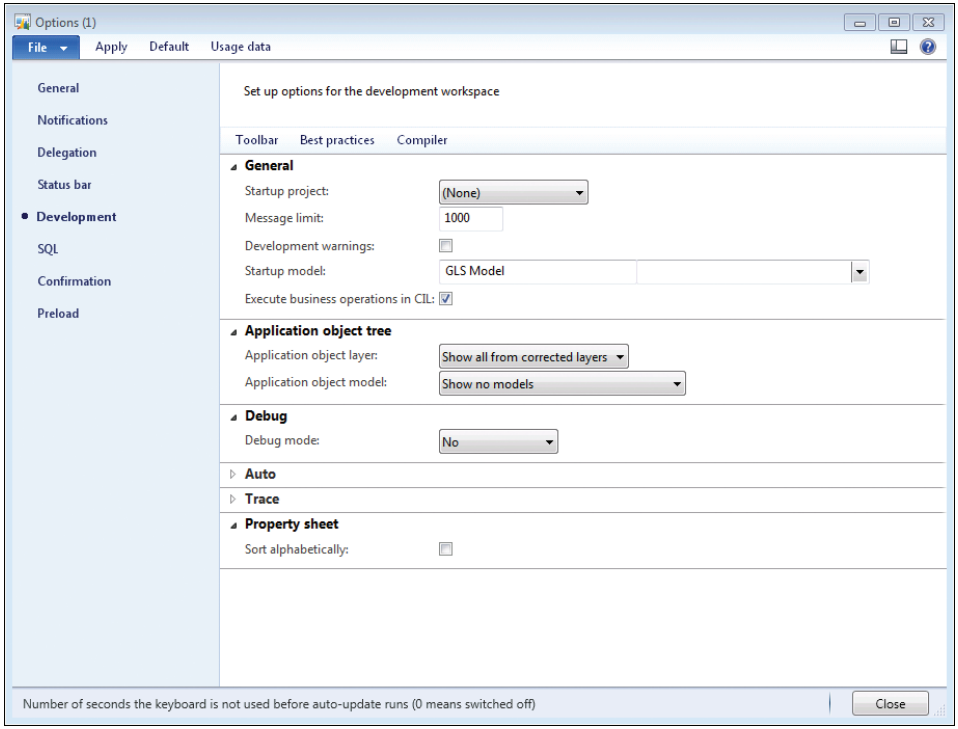


FIGURE 2-1 The Options form, in which development options are specified.

Navigate through the AOT

As the name implies, the AOT is a tree view. The root of the AOT contains the element categories, such as Classes, Tables, and Forms. Some elements are grouped into subcategories to provide a better structure. For example, Tables, Maps, Views, and Extended Data Types are located under Data Dictionary, and all web-related elements are located under Web. Figure 2-2 shows the AOT.

You can navigate through the AOT by using the arrow keys on the keyboard. Pressing the right arrow key expands a node if it has any children.

Elements are arranged alphabetically. Because thousands of elements exist, understanding the naming conventions and adhering to them is important to use the AOT effectively.

All element names in the AOT use the following structure:

<Business area name> + <Functional area> + <Functionality, action performed, or type of content>

With this naming convention, similar elements are placed next to each other. The business area name is also often referred to as the *prefix*. Prefixes are commonly used to indicate the team responsible for an element. For example, in the name *VendPaymReconciliationImport*, the prefix *Vend* is an abbreviation of the business area name (Vendor), *PaymReconciliation* describes the functional area (payment reconciliation), and *Import* lists the action performed (import). The name *CustPaymReconciliationImport* describes a similar functional area and action for the business area Customer.

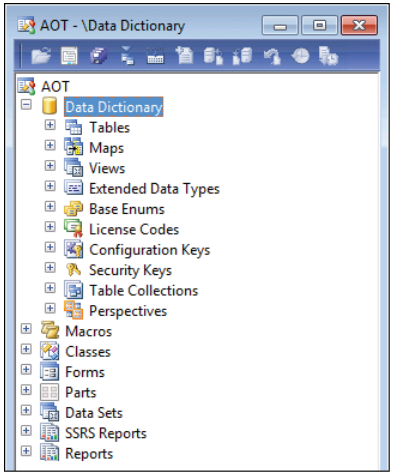


FIGURE 2-2 The AOT.

Tip When building add-on functionality, in addition to following this naming convention, you should add another prefix that uniquely identifies the solution. This additional prefix will help prevent name conflicts if your solution is combined with work from other sources. Consider using a prefix that identifies the company and the solution. For example, if a company called MyCorp is building a payroll system, it could use the prefix *McPR* on all elements added.

Table 2-2 contains a list of the most common prefixes and their descriptions.

TABLE 2-2 Common prefixes.

Prefix	Description
Ax	Microsoft Dynamics AX typed data source
Axd	Microsoft Dynamics AX business document
Asset	Asset management
BOM	Bill of material
COS	Cost accounting
Cust	Customer
Dir	Directory, global address book
EcoRes	Economic resources
HRM/HCM	Human resources
Invent	Inventory management

Prefix	Description
JMG	Shop floor control
KM	Knowledge management
Ledger	General ledger
PBA	Product builder
Prod	Production
Proj	Project
Purch	Purchase
Req	Requirements
Sales	Sales
SMA	Service management
SMM	Sales and marketing management, also called customer relationship management (CRM)
Sys	Application frameworks and development tools
Tax	Tax engine
Vend	Vendor
Web	Web framework
WMS	Warehouse management

Tip When creating new elements, ensure that you follow the recommended naming conventions. Any future development and maintenance will be much easier.

Projects, described in detail later in this chapter, provides an alternative view of the information in the AOT.

Create elements in the AOT

You can create new elements in the AOT by right-clicking the element category node and selecting New <*Element Type*>, as shown in Figure 2-3.

Elements are given automatically generated names when they are created. However, you should replace the default names with new names that conform to the naming convention.

Modify elements in the AOT

Each node in the AOT has a set of properties and either subnodes or X++ code. You can use the property sheet (shown in Figure 2-9) to inspect or modify properties, and you can use the X++ code editor (shown in Figure 2-11) to inspect or modify X++ code.

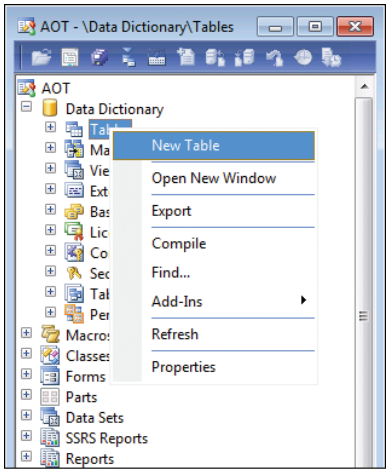


FIGURE 2-3 Creating a new element in the AOT.

The order of the subnodes can play a role in the semantics of the element. For example, the tabs on a form appear in the order in which they are listed in the AOT. You can change the order of nodes by selecting a node and pressing the Alt key while pressing the Up or Down arrow key.

A red vertical line next to a root element name marks it as modified and unsaved, or *dirty*, as shown in Figure 2-4.

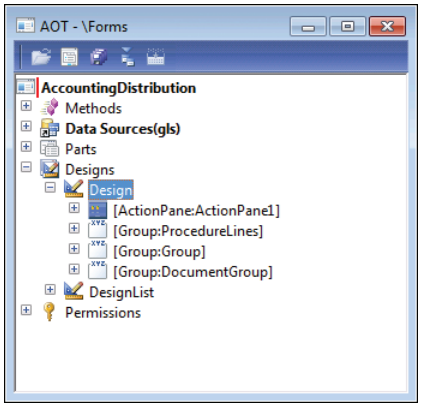


FIGURE 2-4 A dirty element in the AOT, indicated by a vertical line next to the top-level node *AccountingDistribution*.

A dirty element is saved in the following situations:

- The element is executed.
- The developer explicitly invokes the Save or Save All action.

- Autosave takes place. You specify the frequency of autosave in the Options form, which is accessible from the Tools menu.

Refresh elements in the AOT

If several developers modify elements simultaneously in the same installation of Microsoft Dynamics AX, each developer's local elements might not be synchronized with the latest version. To ensure that the local versions of remotely changed elements are updated, an autorefresh thread runs in the background. This autorefresh functionality eventually updates all changes, but you might want to force a refresh explicitly. You do this by right-clicking the element you want to restore and then click Restore. This action refreshes both the on-disk and the in-memory versions of the element.

Typically, the general integrity of what's shown in the AOT is managed automatically, but some operations, such as restoring the application database or reinstalling the application, can lead to inconsistencies that require manual resolution to ensure that the latest elements are used, as follows:

1. Close the Microsoft Dynamics AX client to clear any in-memory elements.
2. Stop the Microsoft Dynamics Server service on the Application Object Server (AOS) to clear any in-memory elements.
3. Delete the application element cache files (*.auc) from the Local Application Data folder (located in "%LocalAppData%") to remove the on-disk elements.

Element actions in the AOT

Each node in the AOT contains a set of available actions. You can access these actions from the context menu, which you can open by right-clicking any node.

Here are two facts to remember about actions:

- The actions that are available depend on the type of node you select.
- You can select multiple nodes and perform actions simultaneously on all the nodes selected.

A frequently used action is Open New Window, which is available for all nodes. It opens a new AOT window with the current node as the root. This action was used to create the screen capture of the *AccountingDistribution* element shown in Figure 2-4. After you open a new AOT window, you can drag elements into the nodes, saving time and effort when you're developing an application.

You can extend the list of available actions on the context menu. You can create custom actions for any element in the AOT by using the features provided by MorphX. In fact, all actions listed on the Add-Ins submenu are implemented in MorphX by using X++ and the MorphX tools.

You can enlist a class as a new add-in by following this procedure:

1. Create a new menu item and give it a meaningful name, a label, and Help text.
2. Set the menu item's *Object Type* property to **Class**.
3. Set the menu item's *Object* property to the name of the class to be invoked by the add-in.
4. Drag the menu item to the *SysContextMenu* menu.
5. If you want the action available only for certain nodes, you need to modify the *verifyItem* method on the *SysContextMenu* class.

Element layers and models in the AOT

When you modify an element from a lower layer, a copy of the element is placed in the current layer and the current model. All elements in the current layer appear in bold type (as shown in Figure 2-5), which makes it easy to recognize changes. For a description of the layer technology, see the "Layers" section in Chapter 21, "Application models."

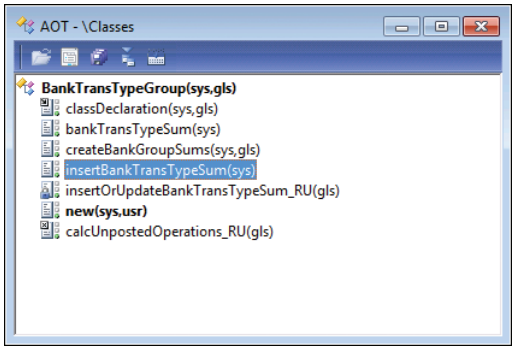


FIGURE 2-5 An element in the AOT that exists in several layers.

You can use the Application object layer and Application object model settings in the Options form to personalize the information shown after the element name in the AOT (see Figure 2-1). Figure 2-5 shows a class with the option set to Show All Layers. As you can see, each method is suffixed with information about the layers in which it is defined, such as *SYS*, *VAR*, and *USR*. If an element exists in several layers, you can right-click it and then click Layers to access its versions from lower layers. It is highly recommended that you use the Show All Layers setting during code upgrade because it provides a visual representation of the layer dimension directly in the AOT.

Projects

For a fully customizable overview of the elements, you can use projects. In a project, you can group and structure elements according to your preference. A project is a powerful alternative to the AOT because you can collect all the elements needed for a feature in one project.

Create a project

You open projects from the AOT by clicking the Project icon on the toolbar. Figure 2-6 shows the Projects window and its *Private* and *Shared* projects nodes.

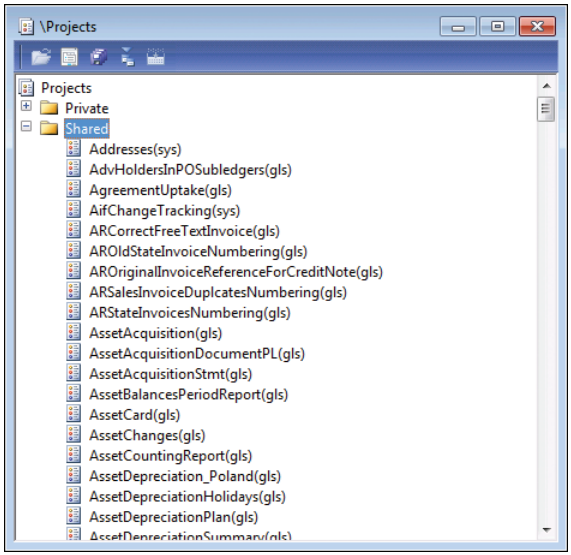


FIGURE 2-6 The Projects window, showing the list of shared projects.

Except for its structure, a project generally behaves like the AOT. Every element in a project is also present in the AOT.

When you create a new project, you must decide whether it should be private or shared among all developers. You can't set access requirements on shared projects. You can make a shared project private (and a private project shared) by dragging it from the shared category into the private category.

Note Central features of Microsoft Dynamics AX 2012 are captured in shared projects to provide an overview of all the elements in a feature. No private projects are included with the application.

You can specify a startup project in the Options form If specified, the chosen project automatically opens when Microsoft Dynamics AX is started.

Automatically generate a project

Projects can be automatically generated in several ways—from using group masks to customizing project types—to make working with them easier. The following sections outline the various ways to generate projects automatically.

Group masks

Groups are folders in a project. When you create a group, you can have its contents be automatically generated by setting the *ProjectGroupType* property (All is an option) and providing a regular expression as the value of the *GroupMask* property. The contents of the group are created automatically and are kept up to date as elements are created, deleted, and renamed. Using group masks ensures that your project is always current, even when elements are created directly in the AOT.

Figure 2-7 shows the *ProjectGroupType* property set to *Classes* and the *GroupMask* property set to *ReleaseUpdate* on a project group. All classes with names containing *ReleaseUpdate* (the prefix for data upgrade scripts) will be included in the project group.

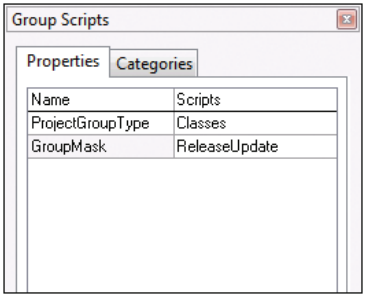


FIGURE 2-7 Property sheet specifying settings for *ProjectGroupType* and *GroupMask*.

Figure 2-8 shows the resulting project when the settings from Figure 2-7 are used.

Filters

You can also generate a project based on a filter. Because all elements in the AOT persist in a database format, you can use a query to filter elements and have the results presented in a project. You create a project filter by clicking the Filter button on the project's toolbar. Depending on the complexity of the query, a project can be generated instantly, or it might take several minutes.

With filters, you can create projects containing elements that meet the following criteria:

- Elements created or modified within the last month
- Elements created or modified by a named user
- Elements from a particular layer

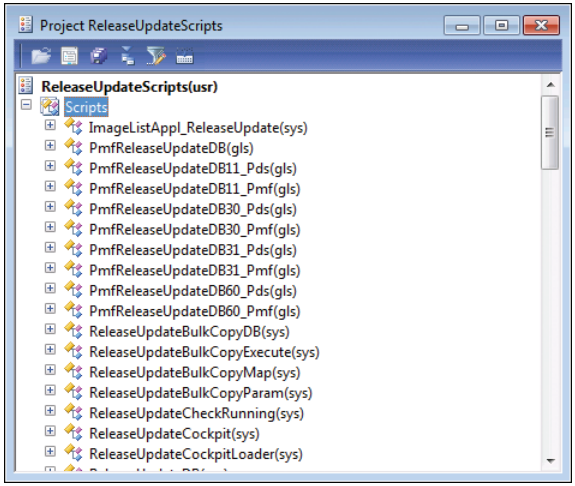


FIGURE 2-8 Project created by using a group mask.

Development tools

Several development tools, such as the Wizard Wizard, produce projects containing elements that the wizard creates. The result of running the Wizard Wizard is a new project that includes a form, a class, and a menu item—all the elements comprising the newly created wizard.

You can also use several other wizards, such as the AIF Document Service Wizard and the Class Wizard, to create projects. To access these wizards, on the Tools menu, click Wizards.

Layer comparison You can compare the elements in one layer with the elements in another layer, which is called the *reference layer*. If an element exists in both layers, and the definitions of the element are different or if the element doesn't exist in the reference layer, the element is added to the resulting project. To compare layers, click Tools > Code Upgrade > Compare Layers.

Upgrade projects When you upgrade from one version of Microsoft Dynamics AX to another or install a new service pack, you need to deal with any new elements that are introduced and existing elements that have been modified. These changes might conflict with customizations you've implemented in a higher layer.

The Create Upgrade Project feature makes a three-way comparison to establish whether an element has any upgrade conflicts. It compares the original version with both the customized version and the updated version. If a conflict is detected, the element is added to the project.

The resulting project provides a list of elements to update based on upgrade conflicts between versions. You can use the Compare tool, described later in this chapter, to see the conflicts in each element. Together, these features provide a cost-effective toolbox to use when upgrading. For more information about code upgrade, see "Microsoft Dynamics AX 2012 White Papers: Code Upgrade" at *http://www.microsoft.com/download/en/details.aspx?id=20864*.

To create an upgrade project, click Tools > Code Upgrade > Detect Code Upgrade Conflicts.

Project types

When you create a new project, you can specify a project type. So far, this chapter has discussed standard projects. The Test project, used to group a set of classes for unit testing, is another specialized project type provided in Microsoft Dynamics AX.

You can create a custom specialized project by creating a new class that extends the *ProjectNode* class. With a specialized project, you can control the structure, icons, and actions available to the project.

Property sheet

Properties are an important part of the metadata system. Each property is a key and value pair. You can use the property sheet to inspect and modify properties of elements.

When the Development Workspace opens, the property sheet is visible by default. If you close it, you can open it again anytime by pressing Alt+Enter or by clicking the Properties button on the toolbar of the Development Workspace. The property sheet automatically updates itself to show properties for any element selected in the AOT. You don't have to open the property sheet manually for each element; you can leave it open and browse the elements. Figure 2-9 shows the property sheet for the *TaxSpec* class. The two columns are the key and value pairs for each property.

Tip Pressing Esc in the property sheet sets the focus back to your origin.

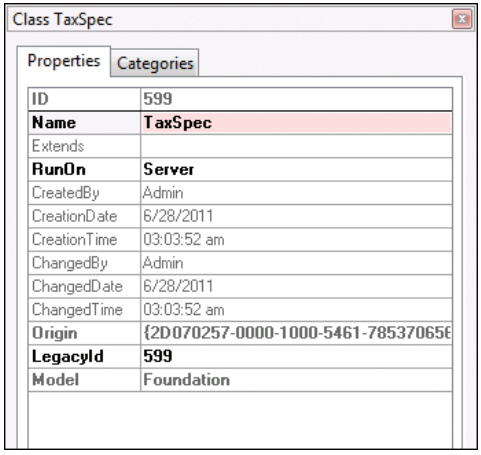


FIGURE 2-9 Property sheet for an element in the AOT.

Figure 2-10 shows the Categories tab for the class shown in Figure 2-9. Here, related properties are categorized. For elements with many properties, this view can make it easier to find the right property.

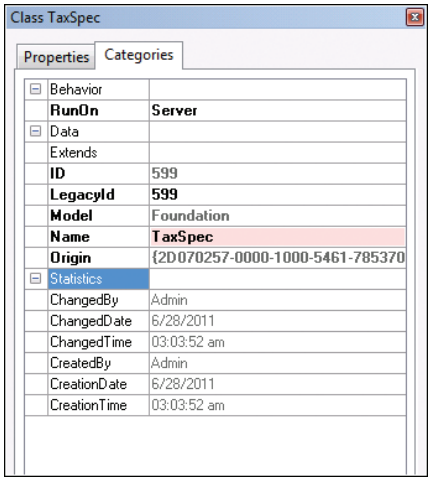


FIGURE 2-10 The Categories tab on the property sheet for an element in the AOT.

Read-only properties appear in gray. Just like files in the file system, elements contain information about who created them and when they were modified. Elements that come from Microsoft all have the same time and user stamps.

The default sort order places related properties near each other. Categories were introduced in an earlier version of Microsoft Dynamics AX to make finding properties easier, but you can also sort properties alphabetically by setting a parameter in the Options form.

You can dock the property sheet on either side of the screen by right-clicking the title bar. Docking ensures that the property sheet is never hidden behind another tool.

X++ code editor

You write all X++ code with the X++ code editor. You open the editor by selecting a node in the AOT and pressing Enter. The editor contains two panes. The left pane shows the methods available, and the right pane shows the X++ code for the selected method, as shown in Figure 2-11.

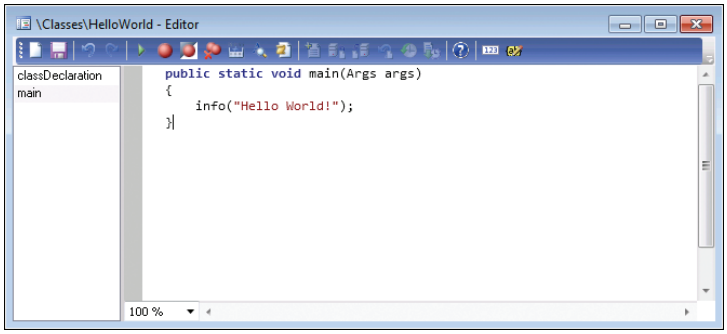


FIGURE 2-11 The X++ code editor.

The X++ code editor is a basic text editor that supports color coding and IntelliSense.

Shortcut keys

Navigation and editing in the X++ code editor use standard shortcuts, as described in Table 2-3. For Microsoft Dynamics AX 2012, some shortcuts differ from those in earlier versions to align with commonly used integrated development environments (IDEs) such as Microsoft Visual Studio.

TABLE 2-3 X++ code editor shortcut keys.

Action	Shortcut	Description
Show Help window	F1	Opens context-sensitive Help for the type or method currently selected in the editor.
Go to next error message	F4	Opens the editor and positions the cursor at the next compilation error, based on the contents of the compiler output window.
Execute current element	F5	Starts the current form, job, or class.
Compile	F7	Compiles the current method.
Toggle a breakpoint	F9	Sets or removes a breakpoint.
Run an editor script	Alt+R	Lists all available editor scripts and lets you select one to execute (such as Send To Mail Recipient).
Open the Label editor	Ctrl+Alt+Spacebar	Opens the Label editor and searches for the selected text.
Go to implementation (drill down in code)	F12	Goes to the implementation of the selected method. This shortcut is highly useful for fast navigation.
Go to the next method	Ctrl+Tab	Sets the focus on the next method in the editor.
Go to the previous method	Ctrl+Shift+Tab	Sets the focus on the previous method in the editor.
Enable block selection	Alt+<mouse select> or Alt+Shift+arrow keys	Selects a block of code. Select the code you want by pressing the Alt key while selecting text with the mouse. Alternatively, hold down Alt and Shift while moving the cursor with the arrow keys.
Cancel selection	Esc	Cancels the current selection.
Delete current selection/line	Ctrl+X	Deletes the current selection or, if nothing is selected, the current line.
Incremental search	Ctrl+I	Starts an incremental search, which marks the first occurrence of the search text as you type it. Pressing Ctrl+I again moves to the next occurrence, and Ctrl+Shift+I moves to the previous occurrence.
Insert XML document header	///	Inserts an XML comment header when you type ///. When done in front of a class or method header, this shortcut prepopulates the XML document with template information relevant to the class or method.
Execute editor script	<*name of script*>+Tab	Runs an editor script when you type the name of an editor script on an empty line in the editor and press Enter. Script names are case sensitive.
Comment selection	Ctrl+E, C	Inserts comment marking for the current selection.
Uncomment selection	Ctrl+E, U	Removes comment marking for the current selection.

Editor scripts

The X++ code editor contains a set of editor scripts that you can invoke by clicking the Script icon on the X++ code editor toolbar or by typing ***<name of script>*+TAB** directly in the editor. Built-in editor scripts provide functionality such as the following:

- Send to mail recipient.
- Send to file.
- Generate code for standard code patterns such as *main*, *construct*, and *parm* methods.
- Open the AOT for the element that owns the method.

Note By generating code, in a matter of minutes you can create a new class with the right constructor method and the right encapsulation of member variables by using *parm* methods. *Parm* methods (*parm* is short for "parameter") are used as simple property getters and setters on classes. Code is generated in accordance with X++ best practices.

Tip To add a main method to a class, add a new method, press Ctrl+A to select all code in the editor tab for the new method, type **main**, and then press the Tab key. This will replace the text in the editor with the standard template for a static main method.

The list of editor scripts is extendable. You can create your own scripts by adding new methods to the *EditorScripts* class.

Label editor

The term *label* in Microsoft Dynamics AX refers to a localizable text resource. Text resources are used throughout the product as messages to the user, form control labels, column headers, Help text in the status bar, captions on forms, and text on web forms, to name just a few places. Labels are localizable, meaning that they can be translated into most languages. Because the space requirement for displaying text resources typically depends on the language, you might fear that the actual user interface must be manually localized as well. However, with IntelliMorph technology, the user interface is dynamically rendered and honors any space requirements imposed by localization.

The technology behind the label system is simple. All text resources are kept in a Unicode-based label file that must have a three-letter identifier. In Microsoft Dynamics AX 2012, the label files are managed in the AOT and distributed using model files. Figure 2-12 shows how the *Label Files* node in the AOT looks with multiple label files and the language en-us.

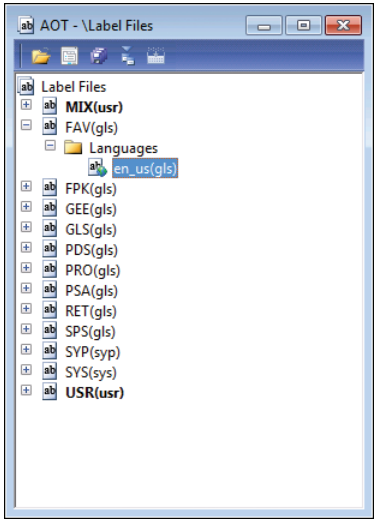


FIGURE 2-12 The *Label Files* node in the AOT.

The underlying source representation is a simple text file following this naming convention:

Ax<Label file identifier><Locale>*.ALD*

The following are two examples, the first showing U.S. English and the second a Danish label file:

Axsysen-us.ALD

Axtstda.ALD

Each text resource in the label file has a 32-bit integer label ID, label text, and an optional label description. The structure of the label file is simple:

@<Label file identifier><Label ID> <Label text>

[Label description]

Figure 2-13 shows an example of a label file.

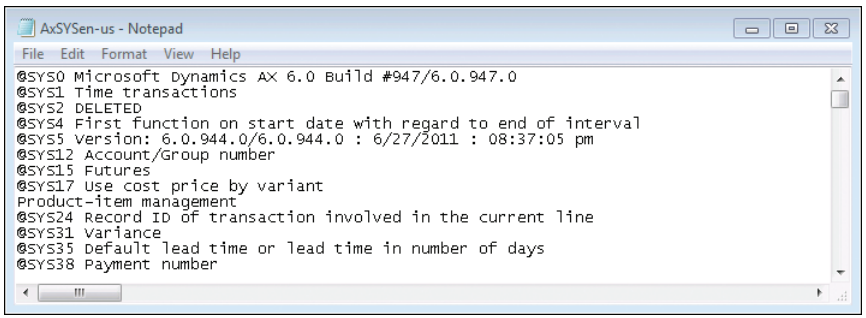


FIGURE 2-13 Label file opened in Windows Notepad showing a few labels from the en-us label file.

This simple structure allows for localization outside of Microsoft Dynamics AX with third-party tools. The AOT provides a set of operations for the label files, including an Export To Label file that can be used to extract a file for external translation.

You can create new label files by using the Label File Wizard, which you access directly from the *Label Files* node in the AOT, or from the Tools menu by pointing to Wizards > Label File Wizard. The wizard guides you through the steps of adding a new label file or a new language to an existing label file. After you run the wizard, the label file is ready to use. If you have an existing .ald file, you can also create the appropriate entry in the AOT by using Create From File on the context menu of the *Label Files* node in the AOT.

Note You can use any combination of three letters when naming a label file, and you can use any label file from any layer. A common misunderstanding is that the label file identifier must match the layer in which it is used. Microsoft Dynamics AX includes a *SYS* layer and a label file named *SYS*; service packs contain a *SYP* layer and a label file named *SYP*. This naming standard was chosen because it is simple, easy to remember, and easy to understand. However, Microsoft Dynamics AX doesn't impose any limitations on the label file name.

Consider the following tips for working with label files:

- When naming a label file, choose a three-letter ID that has a high chance of being unique, such as your company's initials. Don't choose the name of the layer such as *VAR* or *USR*. Eventually, you'll probably merge two separately developed features into the same installation, a task that will be more difficult if the label file names collide.
- When referencing existing labels, feel free to reference labels in the label files provided by Microsoft, but avoid making changes to labels in these label files because they are updated with each new version of Microsoft Dynamics AX.

Create a label

You use the Label editor to create new labels. You can start the Label editor by using any of the following procedures:

- On the Tools menu, point to Label > Label Editor.
- On the X++ code editor toolbar, click the Lookup Label > Text button.
- On text properties in the property sheet, click the Lookup button.

You can use the Label editor (shown in Figure 2-14) to find existing labels. Reusing a label is sometimes preferable to creating a new one. You can create a new label by pressing Ctrl+N or by clicking New.

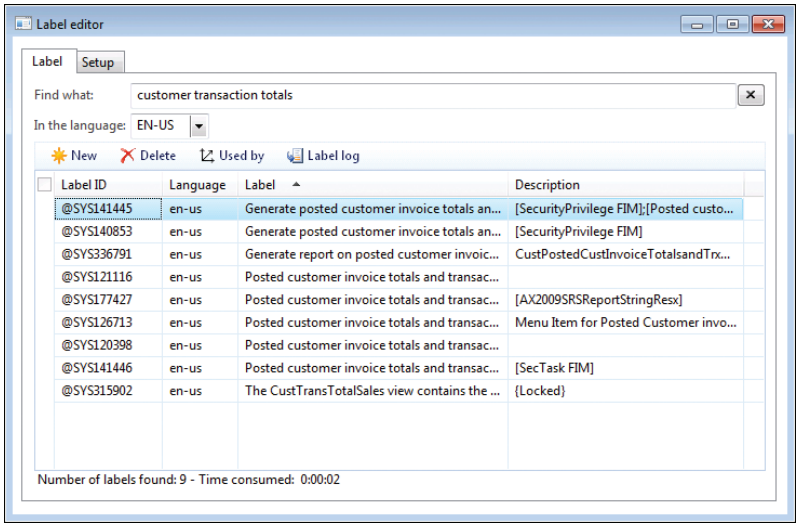


FIGURE 2-14 The Label editor.

In addition to finding and creating new labels, you can also use the Label editor to find out where a label is used. The Label editor also logs any changes to each label.

Consider the following tips when creating and reusing labels:

- When reusing a label, make sure that the label means what you intend it to in all languages. Some words are homonyms (words that have many meanings), and they naturally translate into many different words in other languages. For example, the English word *can* is both a verb and a noun. Use the description column to note the intended meaning of the label.
- When creating a new label, ensure that you use complete sentences or other stand-alone words or phrases. Don't construct complete sentences by concatenating labels with one or two words because the order of words in a sentence differs from one language to another.

Reference labels from X++

In the MorphX design environment, labels are referenced in the format @<LabelFileIdentifier><LabelID>. If you don't want a label reference to be converted automatically to the label text, you can use the *literalStr* function. When a placeholder is needed to display the value of a variable, you can use the *strFmt* function and a string containing %*n*, where *n* is greater than or equal to 1. Placeholders can also be used within labels. The following code shows a few examples:

```
// prints: Time transactions
print "@SYS1";

// prints: @SYS1
print literalStr("@SYS1");
```

```
// prints: Microsoft Dynamics is a Microsoft brand
print strFmt("%1 is a %2 brand", "Microsoft Dynamics", "Microsoft");
pause;
```

The following are some best practices to consider when referencing labels from X++:

- Always create user interface text by using a label. When referencing labels from X++ code, use double quotation marks.
- Never create system text such as file names by using a label. When referencing system text from X++ code, use single quotation marks. You can place system text in macros to make it reusable.

Using single and double quotation marks to differentiate between system text and user interface text allows the Best Practices tool to find and report any hard-coded user interface text. The Best Practices tool is described in depth later in this chapter.

Code compiler

Whenever you make a change to X++ code, you must recompile, just as you would in any other programming language. You start the recompile by pressing F7 in the X++ code editor. Your code also recompiles whenever you close the editor or save changes to an element.

The compiler also produces a list of the following information:

- **Compiler errors** These prevent code from compiling and should be fixed as soon as possible.
- **Compiler warnings** These typically indicate that something is wrong in the implementation. See Table 2-4, later in this section, for a list of compiler warnings. Compiler warnings can and should be addressed. Check-in attempts with compiler warnings are rejected unless specifically allowed in the version control system settings.
- **Tasks (also known as *to-dos*)** The compiler picks up single-line comments that start with *TODO*. These comments can be useful during development for adding reminders, but you should use them only in cases in which implementation can't be completed. For example, you might use a to-do comment when you're waiting for a check-in from another developer. Be careful when using to-do comments to postpone work, and never release code unless they are addressed. For a developer, there is nothing worse than debugging an issue and finding a to-do comment indicating that the issue was already known but overlooked.
- **Best practice deviations** The Best Practices tool carries out more complex validations. For more information, see the "Best Practices tool" later in this chapter.

Note Unlike other languages, X++ requires that you compile only code you've modified, because the intermediate language the compiler produces is persisted along with the X++ code and metadata. Of course, your changes can require other methods consuming your code to be changed and recompiled if, for example, you rename a method or modify its parameters. If the consumers are not recompiled, a run-time error is thrown when they are invoked. This means that you can execute your business application even when compile errors exist, so long as you don't use the code that can't compile. Always ensure that you compile the entire AOT when you consider your changes complete and fix any compilation errors found. If you're changing the class declaration somewhere in a class hierarchy, all classes deriving from the changed class should be recompiled too. This can be achieved using the Compile Forward option under Add-Ins in the context menu for the changed class node.

The Compiler output window provides access to every issue found during compilation, as shown in Figure 2-15. The window presents one list of all relevant errors, warnings, best practices, and tasks. Each type of message can be disabled or enabled by using the respective buttons. Each line in the list contains information about each issue that the compiler detects, a description of the issue, and its location.

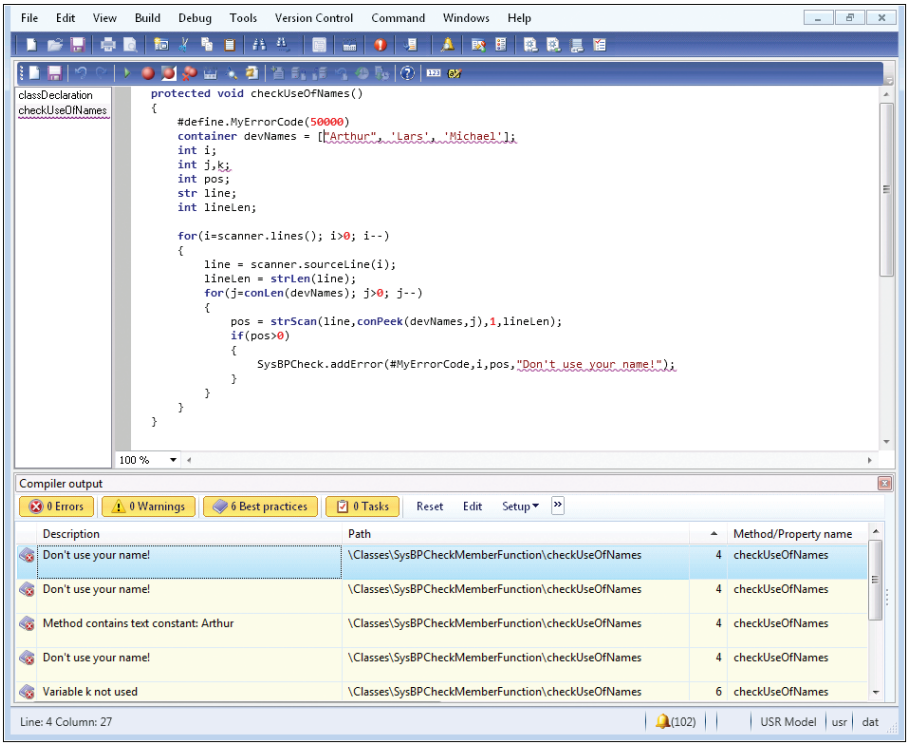


FIGURE 2-15 The powerful combination of the X++ code editor and the Compiler output window.

You can export the contents of the Compiler output window. This capability is useful if you want to share the list of issues with team members. The exported file is an HTML file that can be viewed in Windows Internet Explorer or reimported into the Compiler output window in another Microsoft Dynamics AX session.

In the Compiler output window, click Setup > Compiler to define the types of issues that the compiler should report. Compiler warnings are grouped into four levels, as shown by the examples in Table 2-4. Each level represents a certain level of severity, with 1 being the most critical and 4 being recommended to comply with best practices.

TABLE 2-4 Example compiler warnings.

Warning message	Level
Break statement found outside legal context	1
The new method of a derived class does not call super()	1
The new method of a derived class may not call super()	1
Function never returns a value	1
Not all paths return a value	1
Assignment/comparison loses precision	1
Unreachable code	2
Empty compound statement	3
Class names should start with an upper case letter	4
Member names should start with a lower case letter	4

Best Practices tool

Following Microsoft Dynamics AX best practices when you develop applications has several important benefits:

- You avoid less-than-obvious pitfalls. Following best practices helps you avoid many obstacles, even those that appear only in border scenarios that would otherwise be difficult and time consuming to detect and test. Using best practices allows you to take advantage of the combined experience of Microsoft Dynamics AX expert developers.
- Your learning curve is flattened. When you perform similar tasks in a standard way, you are more comfortable in an unknown area of the application. Consequently, adding new resources to a project is more cost effective, and downstream consumers of the code can make changes more readily.
- You are making a long-term investment. Code that conforms to standards is less likely to require rework during an upgrade process, whether you're upgrading to Microsoft Dynamics AX 2012, installing service packs, or upgrading to future releases.
- You are more likely to ship on time. Most of the problems you face when implementing a solution in Microsoft Dynamics AX have been solved at least once before. Choosing a proven solution results in faster implementation and less regression. You can find solutions to known problems in both the Developer Help section of the SDK and the code base.

The Microsoft Dynamics AX 2012 SDK contains an important discussion about conforming to best practices in Microsoft Dynamics AX. Constructing code that follows proven standards and patterns can't guarantee a project's success, but it minimizes the risk of failure through late, expensive discovery and decreases the long-term maintenance cost. The Microsoft Dynamics AX 2012 SDK is available at *http://msdn.microsoft.com/en-us/library/aa496079.aspx*.

The Best Practices tool is a powerful supplement to the best practices discussion in the SDK. This tool is the MorphX version of a static code analysis tool, similar to FxCop for the Microsoft .NET Framework. The Best Practices tool is embedded in the compiler, and the results are reported in the Compiler output window the same way as other messages from the compilation process.

The purpose of static code analysis is to detect defects and risky coding patterns in the code automatically. The longer a defect exists, the more costly it becomes to fix—a bug found in the design phase is much cheaper to correct than a bug in shipped code running at several customer sites. The Best Practices tool allows any developer to run an analysis of his or her code and application model to ensure that it conforms to a set of predefined rules. Developers can run analysis during development, and they should always do so before implementations are tested. Because an application in Microsoft Dynamics AX is much more than just code, the Best Practices tool also performs static analysis on the metadata—the properties, structures, and relationships that are maintained in the AOT.

The Best Practices tool displays deviations from the best practice rules, as shown in Figure 2-15. Double-clicking a line on the Best Practices tab opens the X++ code editor on the violating line of code or, if the Best Practices violation is related to metadata, it will open the element in an AOT window.

Rules

The Best Practices tool includes about 400 rules, a small subset of the best practices mentioned in the SDK. You can define the best practice rules that you want to run in the Best practice parameters dialog box: on the Tools menu, click > Options > Development, and then click Best Practices.

Note You must set the compiler error level to 4 if you want best practice rule violations to be reported. To turn off the Best Practices tool, on the Tools menu, click Options > Development, and then click Compiler and set the diagnostic level to less than 4.

The best practice rules are divided into categories. By default, all categories are turned on, as shown in Figure 2-16.

The best practice rules are divided into three levels of severity:

- **Errors** The majority of the rules focus on errors. Any check-in attempt with a best practice error is rejected. You must take all errors seriously and fix them as soon as possible.
- **Warnings** Follow a 95/5 rule for warnings. This means that you should treat 95 percent of all warnings as errors; the remaining 5 percent constitute exceptions to the rule. You should provide valid explanations in the design document for all warnings you choose to ignore.

- **Information** In some situations, your implementation might have a side effect that isn't obvious to you or the user (for example, if you assign a value to a variable but you never use the variable again). These are typically reported as information messages.

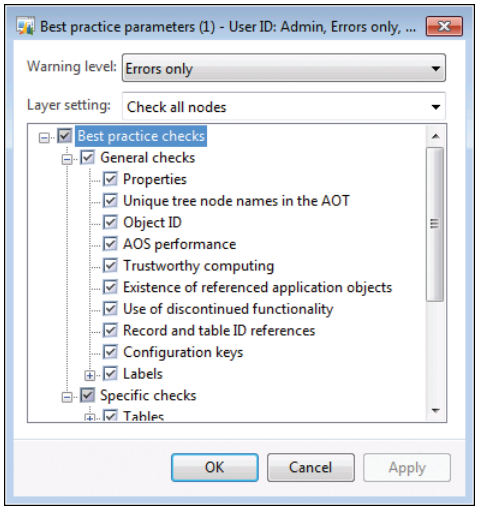


FIGURE 2-16 The Best Practice Parameters dialog box.

Suppress errors and warnings

The Best Practices tool allows you to suppress errors and warnings. A suppressed best practice deviation is reported as information. This gives you a way to identify the deviation as reviewed and accepted. To identify a suppressed error or warning, place a line containing the following text just before the deviation:

```
//BP Deviation Documented
```

Only a small subset of the best practice rules can be suppressed. Use the following guidelines for selecting which rules to suppress:

- **Dangerous API exceptions** When exceptions exist that are impossible to detect automatically, examine each error to ensure the correct implementation. Dangerous application programming interfaces (APIs) are often responsible for such exceptions. A dangerous API is an API that can compromise a system's security when used incorrectly. If a dangerous API is used, a suppressible error is reported. You can use some so-called dangerous APIs when you take certain precautions, such as using code access security (CAS). You can suppress the error after you apply the appropriate mitigations.
- **False positives** About 5 percent of all warnings are false positives and can be suppressed. Note that only warnings caused by actual code can be suppressed this way, not warnings caused by metadata.

After you set up the best practices, the compiler automatically runs the best practices check whenever an element is compiled. The results are displayed on the Best Practices list in the Compiler output dialog box.

Some of the metadata best practice violations can also be suppressed, but the process of suppressing them is different. Instead of adding a comment to the source code, the violation is added to a global list of ignored violations. This list is maintained in the macro named *SysBPCheckIgnore*. This allows for central review of the number of suppressions, which should be kept to a minimum.

Add custom rules

You can use the Best Practices tool to create your own set of rules. The classes used to check for rules are named *SysBPCheck*<Element type>. You call the *init*, *check*, and *dispose* methods once for each node in the AOT for the element being compiled.

One of the most interesting classes is *SysBPCheckMemberFunction*, which is called for each piece of X++ code whether it is a class method, form method, macro, or other method. For example, if developers don't want to include their names in the source code, you can implement a best practice check by creating the following method on the *SysBPCheckMemberFunction* class:

```
protected void checkUseOfNames()
{
    #Define.MyErrorCode(50000)
    container devNames = ['Arthur', 'Lars', 'Michael'];
    int i;
    int j,k;
    int pos;
    str line;
    int lineLen;

    for (i=scanner.lines(); i>0; i--)
    {
        line = scanner.sourceLine(i);
        lineLen = strLen(line);
        for (j=conLen(devNames); j>0; j--)
        {
            pos = strScan(line, conPeek(devNames, j), 1, lineLen);
            if (pos)
            {
                sysBPCheck.addError(#MyErrorCode, i, pos,
                    "Don't use your name!");
            }
        }
    }
}
```

To enlist the rule, make sure to call the preceding method from the *check* method. Compiling this sample code results in the best practice errors shown in Table 2-5.

TABLE 2-5 Best practice errors in *checkUseOfNames*.

Message	Line	Column
Don't use your name!	4	28
Don't use your name!	4	38
Don't use your name!	4	46
Variable k not used	6	11
Method contains text constant: 'Don't use your name!'	20	59

In an actual implementation, names of developers would probably be read from a file. Ensure that you cache the names to prevent the compiler from going to the disk to read the names for each method being compiled.

Note The best practice check also identified that the code contained a variable named *k* that was declared, but never referenced. This is one of the valuable checks ensuring that the code can easily be kept up to date, which helps avoid mistakes. In this case, *k* was not intended for a specific purpose and can be removed.

Debugger

Like most development environments, MorphX features a debugger. The debugger is a stand-alone application, not part of the Microsoft Dynamics AX shell like the rest of the tools mentioned in this chapter. As a stand-alone application, the debugger allows debugging of X++ in any of the following Microsoft Dynamics AX components:

- Microsoft Dynamics AX client
- AOS
- Business Connector (BC.NET)

For other debugging scenarios, such as web services, Microsoft SQL Server Reporting Services (SSRS) reports, and Enterprise Portal, see Chapter 3, "Microsoft Visual Studio tools for Microsoft Dynamics AX."

Enable debugging

For the debugger to start, a breakpoint must be hit when X++ code is executed. You set breakpoints by using the X++ code editor in the Microsoft Dynamics AX Development Workspace. The debugger starts automatically when any component hits a breakpoint.

You must enable debugging for each component as follows:

- In the Development Workspace, on the Tools menu, click Options > Development > Debug, and then select When Breakpoint in the Debug mode list.
- From the AOS, open the Microsoft Dynamics AX Server Configuration Utility under Start > Administrative Tools > Microsoft Dynamics AX 2012 Server Configuration. Create a new configuration, if necessary, and then select the check box Enable Breakpoints to debug X++ code running on this server.
- For Enterprise Portal code that uses the BCPROXY context to run interpreted X++ code, in the Microsoft Dynamics AX Server Configuration Utility, create a new configuration, if necessary, and select the check box Enable Global Breakpoints.

Ensure that you are a member of the local Windows Security Group named Microsoft Dynamics AX Debugging Users. This is normally ensured using setup, but if you did not set up Microsoft Dynamics AX by using your current account, you need to do this manually through Edit Local Users And Groups in the Windows Control Panel. This is necessary to prohibit unauthorized debugging, which could expose sensitive data, provide a security risk, or impose unplanned service disruptions.

Caution It is recommended that you do not enable any of the debugging capabilities in a live environment. If you do, execution will stop when it hits a breakpoint, and the client will stop responding to users. Running the application with debug support enabled also noticeably affects performance.

To set or remove breakpoints, press F9. You can set a breakpoint on any line you want. If you set a breakpoint on a line without an X++ statement, however, the breakpoint will be triggered on the next X++ statement in the method. A breakpoint on the last brace will never be hit.

To enable or disable a breakpoint, press Ctrl+F9. For a list of all breakpoints, press Shift+F9.

Breakpoints are persisted in the *SysBreakpoints* and *SysBreakpointLists* database tables. Each developer has his or her own set of breakpoints. This means that your breakpoints are not cleared when you close Microsoft Dynamics AX and that other Microsoft Dynamics AX components can access them and break where you want them to.

Debugger user interface

The main window in the debugger initially shows the point in the code where a breakpoint was hit. You can control execution one step at a time while inspecting variables and other aspects of the code. Figure 2-17 shows the debugger opened to a breakpoint with all the windows enabled.

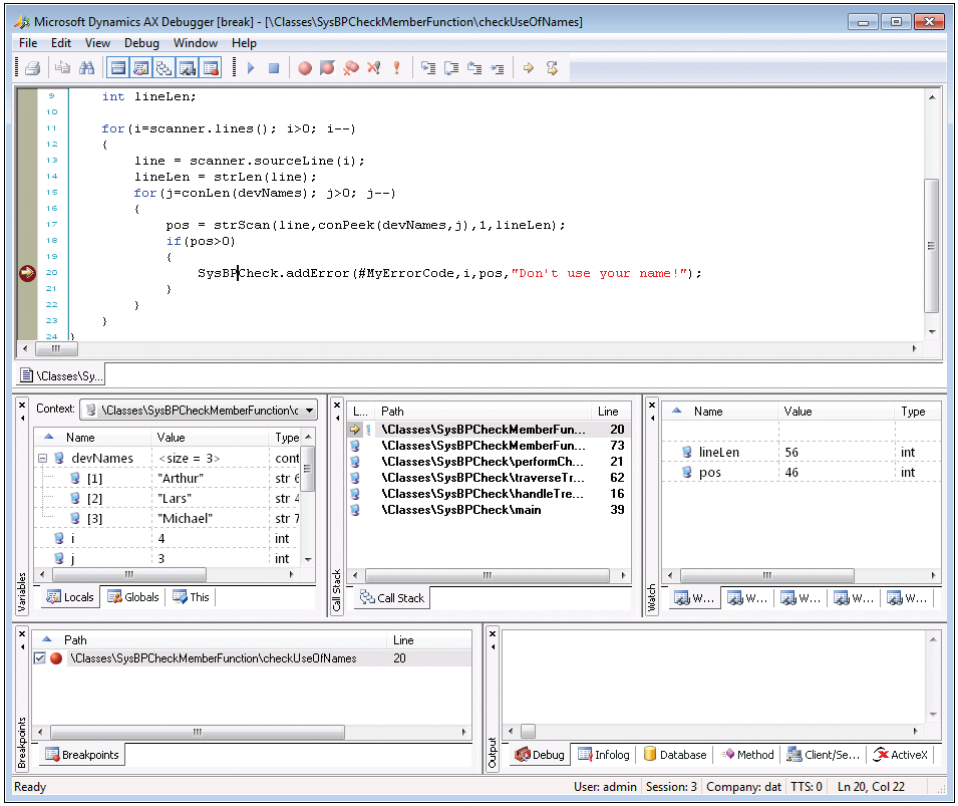


FIGURE 2-17 Debugger with all windows enabled.

Table 2-6 describes the debugger's various windows and some of its other features.

TABLE 2-6 Debugger user interface (UI) elements.

Debugger element	Description
Code window	Shows the current X++ code. Each variable has a ScreenTip that reveals its value. You can drag the next-statement pointer in the left margin. This pointer is particularly useful if the execution path isn't what you expected or if you want to repeat a step.
Variables window	Shows local, global, and member variables, along with their names, values, and types. Local variables are variables in scope at the current execution point. Global variables are the global classes that are always instantiated: Appl, Infolog, ClassFactory, and VersionControl. Member variables make sense only on classes, and they show the class member variables. If a variable is changed as you step through execution, it is marked in red. Each variable is associated with a client or server icon. You can modify the value of a variable by double-clicking the value.

Call Stack window	Shows the code path followed to arrive at a particular execution point. Clicking a line in the Call Stack window opens the code in the Code window and updates the local Variables window. A client or server icon indicates the tier on which the code is executed.
Watch window	Shows the name, value, and type of the variables. Five different Watch windows are available. You can use these to group the variables you're watching in the way that you prefer. You can use this window to inspect variables without the scope limitations of the Variables window. You can drag a variable here from the Code window or the Variables window.
Breakpoints window	Lists all your breakpoints. You can delete, enable, and disable the breakpoints through this window.
Output window	Shows the traces that are enabled and the output sent to the Infolog application framework, which is introduced in Chapter 5, "Designing the user experience." The Output window includes the following pages: ■ **Debug** You can instrument your X++ code to trace to this page by using the *printDebug* static method on the *Debug* class. ■ **Infolog** This page contains messages in the queue for the Infolog. ■ **Database, Client/Server, and ActiveX Trace** Any traces enabled on the Development tab in the Options form appear on these pages.
Status bar window	Provides the following important context information: ■ **Current user** The ID of the user who is logged on to the system. This information is especially useful when you are debugging incoming web requests. ■ **Current session** The ID of the session on the AOS. ■ **Current company accounts** The ID of the current company accounts. ■ **Transaction level** The current transaction level. When it reaches zero, the transaction is committed.

Tip As a developer, you can provide more information in the value field for your classes than what is provided by default. The defaults for classes are *New* and *Null*. You can change the defaults by overriding the *toString* method. If your class doesn't explicitly extend the *object* (the base class of all classes), you must add a new method named *toString*, returning *str* and taking no parameters, to implement this functionality.

Debugger shortcut keys

Table 2-7 lists the most important shortcut keys available in the debugger.

TABLE 2-7 Debugger shortcut keys.

Action	Shortcut	Description
Run	F5	Continue execution
Stop debugging	Shift+F5	Break execution
Step over	F10	Step over next statement
Run to cursor	Ctrl+F10	Continue execution but break at the cursor's position
Step into	F11	Step into next statement
Step out	Shift+F11	Step out of method
Toggle breakpoint	Shift+F9	Insert or remove breakpoint
Variables window	Ctrl+Alt+V	Open or close the Variables window
Call Stack window	Ctrl+Alt+C	Open or close the Call Stack window
Watch window	Ctrl+Alt+W	Open or close the Watch window
Breakpoints window	Ctrl+Alt+B	Open or close the Breakpoints window
Output window	Ctrl+Alt+O	Open or close the Output window

Reverse Engineering tool

You can generate Visio models from existing metadata. Considering the amount of metadata available in Microsoft Dynamics AX 2012 (more than 50,000 elements and more than 18 million lines of text when exported), it's practically impossible to get a clear view of how the elements relate to each other just by using the AOT. The Reverse Engineering tool is a great aid when you need to visualize metadata.

Note You must have Visio 2007 or later installed to use the Reverse Engineering tool.

The Reverse Engineering tool can generate a Unified Modeling Language (UML) data model, a UML object model, or an entity relationship data model, including all elements from a private or shared project. To open the tool, in the Projects window, right-click a project or a perspective, point to Add-Ins > Reverse Engineer. You can also open the tool by selecting Reverse Engineer from the Tools menu. In the dialog box shown in Figure 2-18, you must specify a file name and model type.

When you click OK, the tool uses the metadata for all elements in the project to generate a Visio document that opens automatically. You can drag elements from the Visio Model Explorer onto the drawing surface, which is initially blank. Any relationship between two elements is automatically shown.

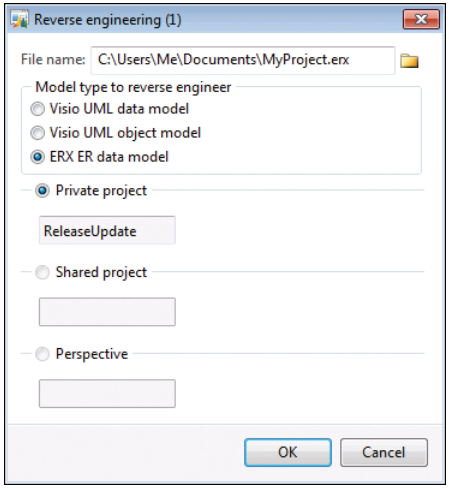


FIGURE 2-18 The Reverse Engineering dialog box.

UML data model

When generating a UML data model, the Reverse Engineering tool looks for tables in the project. The UML model contains a class for each table and view in the project and its attributes and associations. Figure 2-19 shows a class diagram with the *CustTable* (customers), *InventTable* (inventory items), *SalesTable* (sales order header), and *SalesLine* (sales order line) tables. To simplify the diagram, some attributes have been removed.

The UML model also contains referenced tables and all extended data types, base enumerations, and X++ data types. You can include these items in your diagrams without having to run the Reverse Engineering tool again.

Fields in Microsoft Dynamics AX are generated as UML attributes. All attributes are marked as public to reflect the nature of fields in Microsoft Dynamics AX. Each attribute also shows the type. The primary key field is underlined. If a field is a part of one or more indexes, the field name is prefixed with the names of the indexes; if the index is unique, the index name is noted in brackets.

Relationships in Microsoft Dynamics AX are generated as UML associations. The *Aggregation* property of the association is set based on two conditions in metadata:

- If the relationship is validating (the *Validate* property is set to *Yes*), the *Aggregation* property is set to *Shared*. This is also known as a UML aggregation, represented by a white diamond.
- If a cascading delete action exists between the two tables, a composite association is added to the model. A cascading delete action ties the lifespan of two or more tables and is represented by a black diamond.

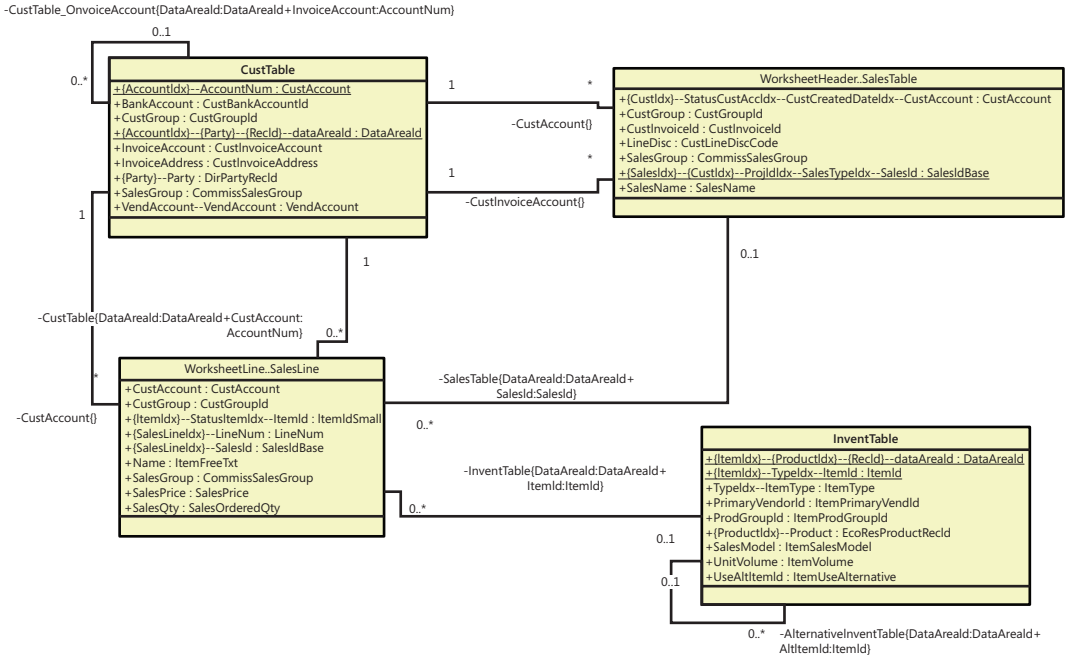


FIGURE 2-19 UML data model diagram.

The name of an association endpoint is the name of the Microsoft Dynamics AX relationship. The names and types of all fields in the relationship appear in brackets.

UML object model

When generating an object model, the Reverse Engineering tool looks for Microsoft Dynamics AX classes, tables, and interfaces in the project. The UML model contains a class for each Microsoft Dynamics AX table and class in the project and an interface for each Microsoft Dynamics AX interface in the project. The UML model also contains attributes and operations, including return types, parameters, and the types of the parameters. Figure 2-20 shows an object model of the most important *RunBase* and *Batch* classes and interfaces in Microsoft Dynamics AX. To simplify the view, some attributes and operations have been removed and operation parameters are suppressed.

The UML model also contains referenced classes, tables, and all extended data types, base enumerations, and X++ data types. You can include these elements in your diagrams without having to run the Reverse Engineering tool again.

Fields and member variables in Microsoft Dynamics AX are generated as UML attributes. All fields are generated as public attributes, whereas member variables are generated as protected attributes. Each attribute also shows the type. Methods are generated as UML operations, including return type, parameters, and the types of the parameters.

The Reverse Engineering tool also picks up any generalizations (classes extending other classes), realizations (classes implementing interfaces), and associations (classes using each other). The associations are limited to references in member variables.

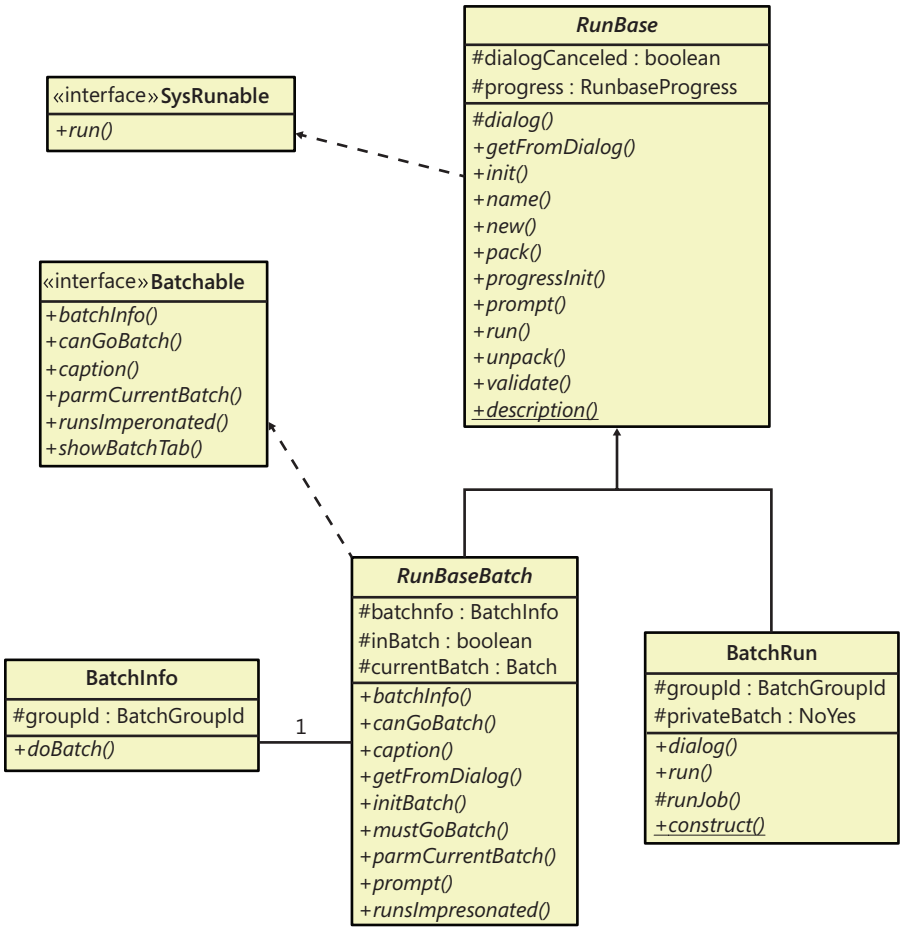


FIGURE 2-20 UML object model diagram.

Note To get the names of operation parameters, you must reverse-engineer in debug mode. The names are read from metadata only and placed into the stack when in debug mode. To enable debug mode, on the Development tab of the Options form, select When Breakpoint in the Debug Mode list.

Entity relationship data model

When generating an entity relationship data model, the Reverse Engineering tool looks for tables and views in the project. The entity relationship model contains an entity type for each AOT table in the project and attributes for the fields in each table. Figure 2-21 shows an Entity Relationship Diagram (ERD) for the tables *HcmBenefit (Benefit)*, *HcmBenefitOption (Benefit option), HcmBenefitType (Benefit type,)* and *HcmBenefitPlan (Benefit plan)*.

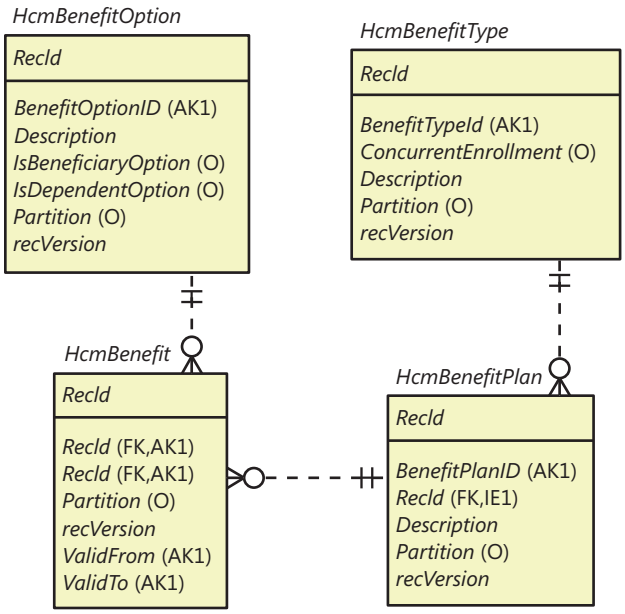


FIGURE 2-21 ERD using IDEF1X notation.

Fields in Microsoft Dynamics AX are generated as entity relationship columns. Columns can be foreign key (FK), alternate key (AK), inversion entry (IE), and optional (O). A foreign key column is used to identify a record in another table, an alternate key uniquely identifies a record in the current table, an inversion entry identifies zero or more records in the current table (these are typical of the fields in nonunique indexes), and optional columns don't require a value.

Relationships in Microsoft Dynamics AX are generated as entity relationships. The *EntityRelationshipRole* property of the relationship in Microsoft Dynamics AX is used as the foreign key role name of the relation in the entity relationship data model.

Note The Reverse Engineering tool produces an ERX file. To work with the generated file in Visio, do the following: In Visio, create a new Database Model Diagram, and then, on the select Database menu, point to Import > Import Erwin ERX file. Afterward, you can drag relevant tables from the Tables And Views pane (available from the Database menu) to the diagram canvas.

Table Browser tool

The Table Browser tool is a small, helpful tool that can be used in numerous scenarios. You can browse and maintain the records in a table without having to build a user interface. This tool is useful when you're debugging, validating data models, and modifying or cleaning up data, to name just a few uses.

To access the Table Browser tool, right-click any of the following types of items in the AOT, and then point to Add-Ins > Table Browser:

- Tables
- Tables listed as data sources in forms, queries, and data sets
- System tables listed in the AOT under System Documentation\Tables

Note The Table Browser tool is implemented in X++. You can find it in the AOT under the name *SysTableBrowser*. It is a good example of how to bind the data source to a table at run time.

Figure 2-22 shows the Table Browser tool when started from the *CustTrans* table. In addition to the querying, sorting, and filtering capabilities provided by the grid control, you can type an SQL *SELECT* statement directly into the form using X++ *SELECT* statement syntax and see a visual display of the result set. This tool is a great way to test complex *SELECT* statements. It fully supports grouping, sorting, aggregation, and field lists.

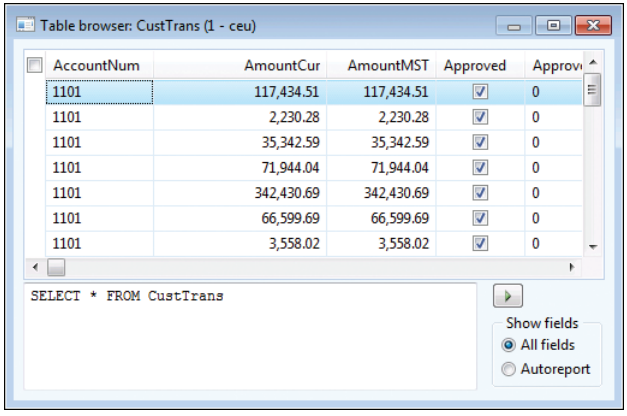


FIGURE 2-22 The Table Browser tool showing the contents of the *CustTrans* table demo data.

You can also choose to see only the fields from the auto-report field group. These fields are printed in a report when the user clicks Print in a form with this table as a data source. Typically, these fields hold the most interesting information. This option can make it easier to find the values you're looking for in tables with many fields.

Note The Table Browser tool is just a standard form that uses IntelliMorph. It can't display fields for which the *visible* property is set to *No* or fields that the current user doesn't have access to.

Find tool

Search is everything, and the size of Microsoft Dynamics AX applications calls for a powerful and effective search tool.

Tip You can use the Find tool to search for an example of how to use an API. Real examples can complement the examples found in the documentation.

You can start the Find tool, shown in Figure 2-23, from any node in the AOT by pressing Ctrl+F or by clicking Find on the context menu. The Find tool supports multiple selections in the AOT.

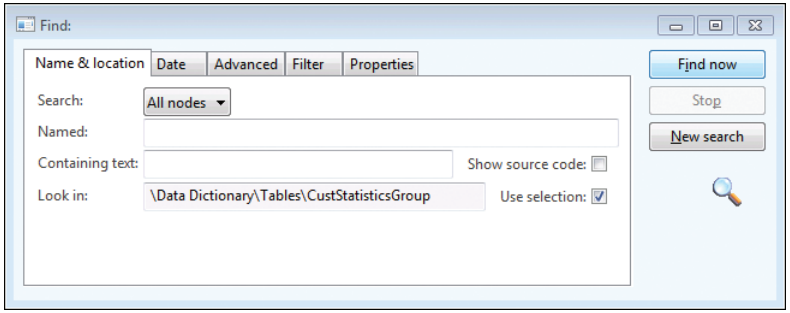


FIGURE 2-23 The Find tool.

On the Name & Location tab, you define what you're searching for and where to look:

- In Search, the menu options are Methods and All Nodes. If you choose All Nodes, the Properties tab appears.
- The Named box limits the search to nodes with the name you specify.
- The Containing box specifies the text to look for in the method, expressed as a regular expression.
- If you select the Show Source Code check box, results include a snippet of source code containing the match, making it easier to browse the results.

By default, the Find tool searches the node (and its subnodes) selected in the AOT. If you change focus in the AOT while the Find tool is open, the Look In value is updated. This is quite useful if you want to search several nodes using the same criterion. You can disable this behavior by clearing the Use Selection check box.

On the Date tab, you specify additional ranges for your search, such as Modified Date and Modified By.

On the Advanced tab, you can specify more advanced settings for your search, such as the layer to search, the size range of elements, the type of element, and the tier on which the element is set to run.

On the Filter tab, shown in Figure 2-24, you can write a more complex query by using X++ and type libraries. The code in the Source text box is the body of a method with the following profile:

```
boolean FilterMethod(str _treeNodeName,
                     str _treeNodeSource,
                     XRefPath _path,
                     ClassRunMode _runMode)
```

The example in Figure 2-24 uses the class *SysScannerClass* to find any occurrence of the *ttsAbort* X++ keyword. The scanner is primarily used to pass tokens into the parser during compilation. Here, however, it detects the use of a particular keyword. This tool is more accurate (though slower) than using a regular expression because X++ comments don't produce tokens.

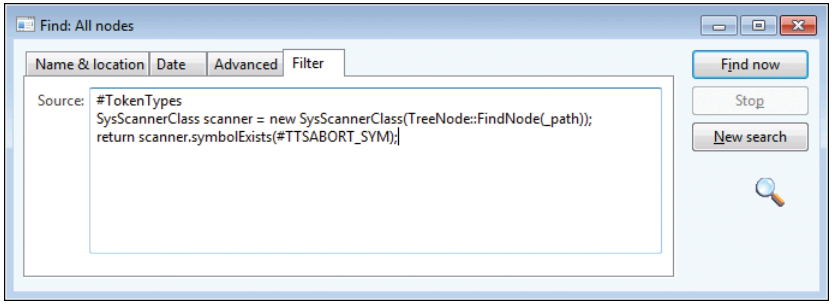


FIGURE 2-24 Filtering in the Find tool.

The Properties tab appears when All Nodes is selected in the Search list. You can specify a search range for any property. Leaving the range blank for a property is a powerful setting when you want to inspect properties: it matches all nodes, and the property value is added as a column in the results, as shown in Figure 2-25. The search begins when you click Find Now. The results appear at the bottom of the dialog box as they are found.

Double-clicking any line in the result set opens the X++ code editor and sets the focus on the code example that matches. When you right-click the lines in the result set, a context menu containing the Add-Ins menu opens.

Compare tool

Several versions of the same element typically exist. These versions might emanate from various layers or revisions in version control, or they could be modified versions that exist in memory. Microsoft Dynamics AX has a built-in Compare tool that highlights any differences between two versions of an element.

The comparison shows changes to elements, which can be modified in three ways:

- A metadata property can be changed.
- X++ code can be changed.
- The order of subnodes can be changed, such as the order of tabs on a form.

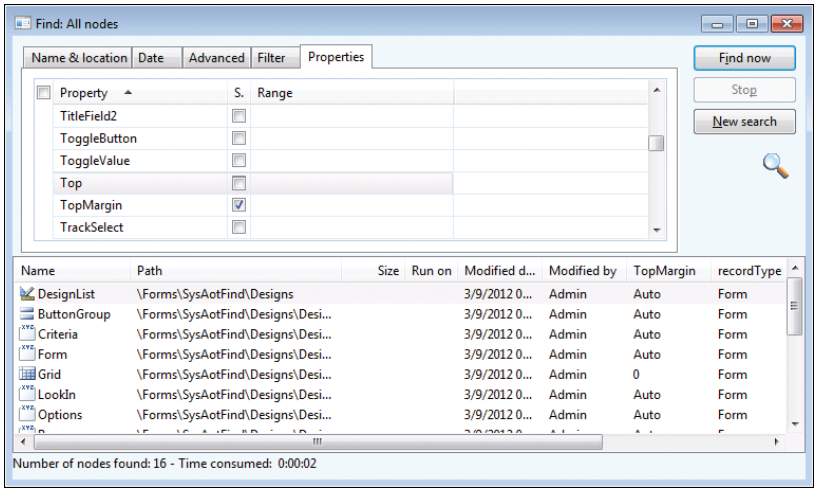


FIGURE 2-25 Search results in the Find tool.

Start the Compare tool

To open the Compare tool, right-click an element, and then click Compare. A dialog box opens where you can select the versions of the element you want to compare, as shown in Figure 2-26.

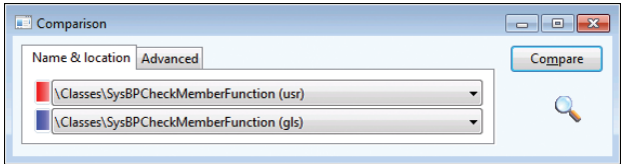


FIGURE 2-26 The Comparison dialog box.

The versions to choose from come from many sources. The following is a list of all possible types of versions:

- **Standard layered version types** These include *SYS*, *SYP*, *GLS*, *GLP*, *FPK*, *FPP*, *SLN*, *SLP*, *ISV*, *ISP*, *VAR*, *VAP*, *CUS*, *CUP*, *USR*, and *USP*.
- **Old layered version types (old *SYS*, old *SYP*, and so on)** If a baseline model store is present, elements from the files are available here. This allows you to compare an older version of an element with its latest version. For more information about layers and the baseline model store, see Chapter 21.

- **Version control revisions (Version 1, Version 2, and so on)** You can retrieve any revision of an element from the version control system individually and use it for comparison. The version control system is explained later in this chapter.
- **Best practice washed version (Washed)** A few simple best practice issues can be resolved automatically by a best practice "wash." Selecting the washed version shows you how your implementation differs from best practices. To get the full benefit of this, select the Case Sensitive check box on the Advanced tab.
- **Export/import file (XPO)** Before you import elements, you can compare them with existing elements (which will be overwritten during import). You can use the Compare tool during the import process (Command > Import) by selecting the Show Details check box in the Import dialog box and right-clicking any elements that appear in bold. Objects in bold already exist in the application.
- **Upgraded version (Upgraded)** MorphX can automatically create a proposal for how a class should be upgraded. The requirement for upgrading a class arises during a version upgrade. The Create Upgrade Project step in the Upgrade Checklist automatically detects customized classes that conflict with new versions of the classes. A class is conflicting if you've changed the original version of the class, and the publisher of the class has also changed the original version. MorphX constructs the proposal by merging your changes with the publisher's changes to the class. MorphX requires access to all three versions of the class—the original version in the baseline model store, a version with your changes in the current layer in the baseline model store, and a version with the publisher's changes in the same layer as the original. The installation program ensures that the right versions are available in the right places during an upgrade. Conflict resolution is shown in Figure 2-27.

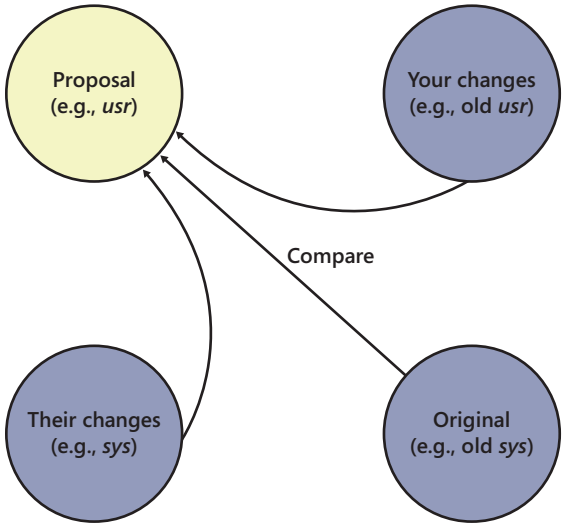


FIGURE 2-27 How the upgraded version proposal is created.

Note You can also compare two different elements. To do this, select two elements in the AOT, right-click, point to Add-Ins, and then click Compare.

Figure 2-28 shows the Advanced tab, on which you can specify comparison options.

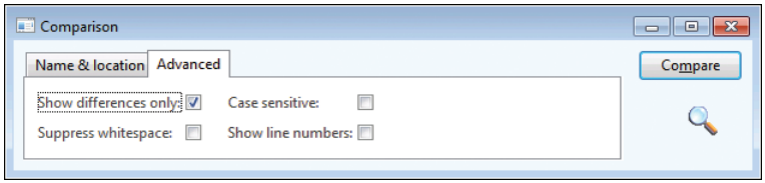


FIGURE 2-28 Comparison options on the Advanced tab.

The following list describes the comparison options shown in Figure 2-28:

- **Show Differences Only** All equal nodes are suppressed from the view, making it easier to find the changed nodes. This option is selected by default.
- **Suppress Whitespace** White space, such as spaces and tabs, is suppressed into a single space during the comparison. The Compare tool can ignore the amount of white space, just as the compiler does. This option is selected by default.
- **Case Sensitive** Because X++ is not case sensitive, the Compare tool is also not case sensitive by default. In certain scenarios, case sensitivity is required and must be enabled, such as when you're using the best practice wash feature mentioned earlier in this section. This option is cleared by default.
- **Show Line Numbers** The Compare tool can add line numbers to all X++ code that is displayed. This option is cleared by default but can be useful during an upgrade of large chunks of code.

Use the Compare tool

After you choose elements and set parameters, start the comparison by clicking Compare. Results are displayed in a three-pane dialog box, as shown in Figure 2-29. The top pane contains the elements and options that you selected, the left pane displays a tree structure resembling the AOT, and the right pane shows details that correspond to the item selected in the tree.

Color-coded icons in the tree structure indicate how each node has changed. A red or blue check mark indicates that the node exists only in a particular version. Red corresponds to the *SYS* layer, and blue corresponds to the old *SYS* layer. A gray check mark indicates that the nodes are identical but one or more subnodes are different. A not-equal-to symbol (≠) on a red and blue background indicates that the nodes are different in the two versions.

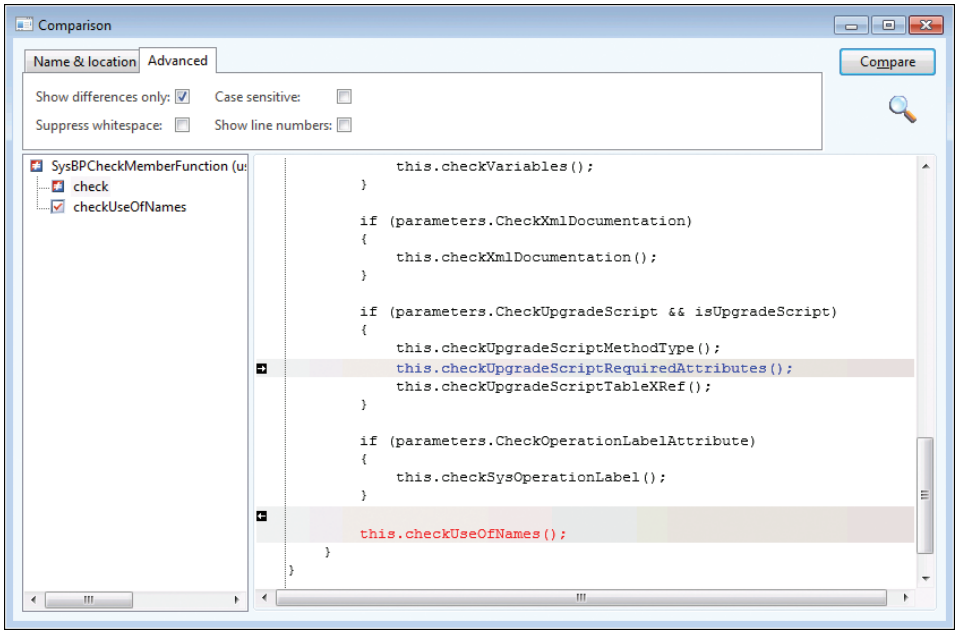


FIGURE 2-29 Comparison results.

Note Each node in the tree view has a context menu that provides access to the Add-Ins submenu and the Open New Window option. The Open New Window option provides an AOT view of any element, including elements in old layers.

Details about the differences are shown in the right pane. Color coding is also used in this pane to highlight differences the same way that it is in the tree structure. If an element is editable, small action icons appear. These icons allow you to make changes to code, metadata, and nodes, which can save you time when performing an upgrade. A right or left arrow removes or adds the difference, and a bent arrow moves the difference to another position. These arrows always come in pairs, so you can see where the difference is moved to and from. If a version control system is in use, an element is editable if it is from the current layer and is checked out.

Compare APIs

Although Microsoft Dynamics AX provides the comparison functionality for development purposes only, the comparison functionality can be reused for other tasks. You can use the available APIs to compare and present differences in the tree structure or text representation of any type of entity.

The *Tutorial_CompareContextProvider* class shows how simple it is to compare business data by using these APIs and present it by using the Compare tool. The tutorial consists of two parts:

- ***Tutorial_Comparable*** This class implements the *SysComparable* interface. Basically, it creates a text representation of a customer.

- ***Tutorial_CompareContextProvider*** This class implements the *SysCompareContextProvider* interface. It provides the context for comparison. For example, it creates a *Tutorial_Comparable* object for each customer, sets the default comparison options, and handles context menus.

Figure 2-30 shows a comparison of two customers, the result of running the tutorial.

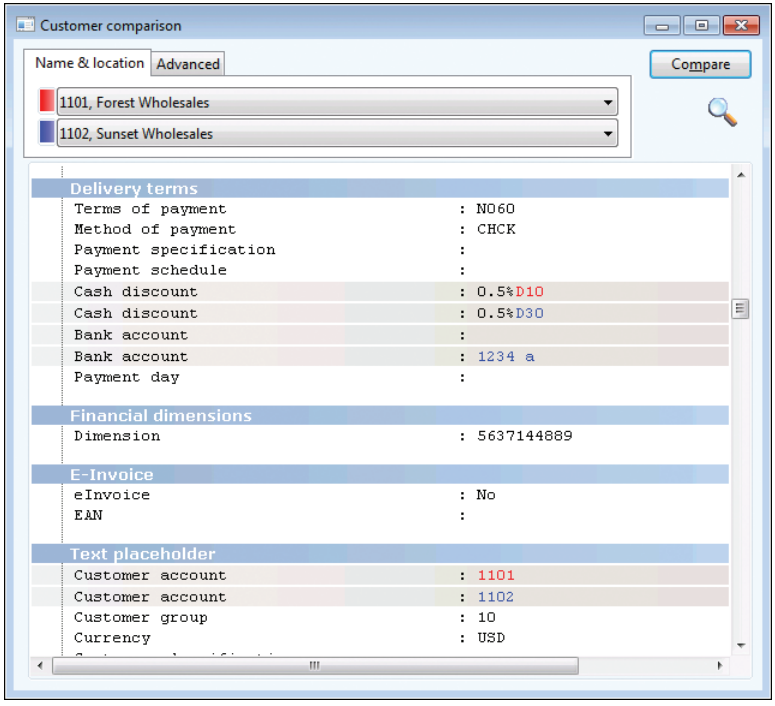


FIGURE 2-30 The result of comparing two customers using the Compare API.

You can also use the line-by-line comparison functionality directly in X++. The static *run* method on the *SysCompareText* class, shown in the following code, takes two strings as parameters and returns a container that highlights differences in the two strings. You can also use a set of optional parameters to control the comparison.

```
public static container run(str _t1,
      str _t2,
      boolean _caseSensitive = false,
      boolean _suppressWhiteSpace = true,
      boolean _lineNumbers  = false,
      boolean _singleLine  = false,
       boolean _alternateLines = false)
```

Cross-Reference tool

The concept of cross-references in Microsoft Dynamics AX is simple. If an element uses another element, the reference is recorded. With cross-references, you can determine which elements a particular element uses and which elements other elements are using. Microsoft Dynamics AX provides the Cross-Reference tool for accessing and managing cross-reference information.

Here are a couple of typical scenarios for using the Cross-Reference tool:

- You want to find usage examples. If the product documentation doesn't help, you can use the Cross-Reference tool to find real implementation examples.
- You need to perform an impact analysis. If you're changing an element, you need to know which other elements are affected by your change.

You must update the Cross-Reference tool regularly to ensure accuracy. The update typically takes several hours. The footprint in a database is about 1.5 GB for a standard application.

To update the Cross-Reference tool, on the Tools menu, point to > Cross-Reference > Periodic > Update. Updating the Cross-Reference tool also compiles the entire AOT because the compiler emits cross-reference information.

Tip Keeping the Cross-Reference tool up to date is important if you want its information to be reliable. If you work in a shared development environment, you share cross-Reference information with your team members. Updating the Cross-Reference tool nightly is a good approach for a shared environment. If you work in a local development environment, you can keep the Cross-Reference tool up to date by enabling cross-referencing when compiling. This option slows down compilation, however. Another option is to update cross-references manually for the elements in a project. To do so, right-click the project and point to Add-Ins > Cross-Reference > Update.

In addition to the main cross-reference information, two smaller cross-reference subsystems exist:

- **Data model** Stores information about relationships between tables. It is primarily used by the query form and the Reverse Engineering tool.
- **Type hierarchy** Stores information about class and data type inheritance.

For more information about these subsystems and the tools that rely on them, see the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/aa496079.aspx*).

The cross-reference information the Cross-Reference tool collects is quite comprehensive. You can find the complete list of cross-referenced elements by opening the AOT, expanding the *System Documentation* node, and clicking Enums and then xRefKind.

When the Cross-Reference tool is updating, it scans all metadata and X++ code for references to elements of the kinds listed here.

Tip It's a good idea to use intrinsic functions when referring to elements in X++ code. An intrinsic function can evaluate to either an element name or an ID. The intrinsic functions are named <Element type>*Str* or <Element type>*Num*, respectively. Using intrinsic functions provides two benefits: you have compile-time verification that the element you reference actually exists, and the reference is picked up by the Cross-Reference tool. Also, there is no run-time overhead. An example follows:

```
// Prints ID of MyClass, such as 50001
print classNum(myClass);

// Prints "MyClass"
print classStr(myClass);

// No compile check or cross-reference
print "MyClass";
```

For more information about intrinsic functions, see Chapter 20, "Reflection."

To access usage information, right-click any element in the AOT and point to Add-Ins > Cross-Reference > Used By. If the option isn't available, either the element isn't used or the cross-reference hasn't been updated.

Figure 2-31 shows where the *prompt* method is used on the *RunBaseBatch* class.

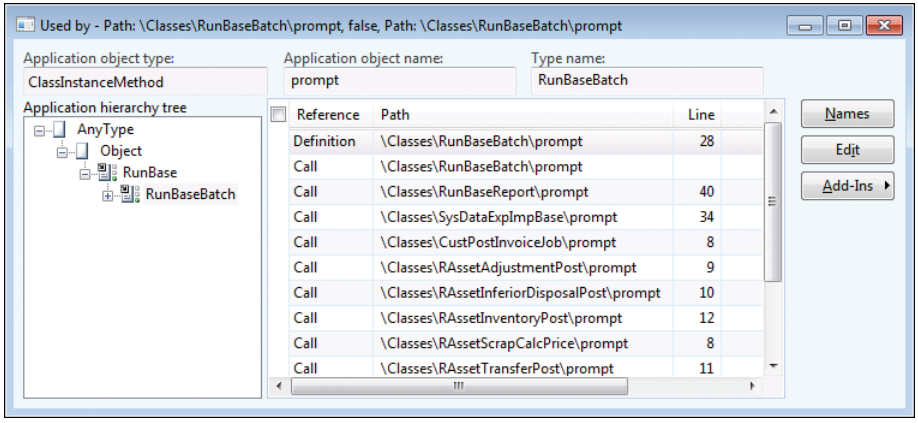


FIGURE 2-31 The Cross-Reference tool, showing where *RunBaseBatch.prompt* is used

When you view cross-references for a class method, the Application hierarchy tree is visible, so that you can see whether the same method is used on a parent or subclass. For types that don't support inheritance, the Application hierarchy tree is hidden.

Version control

The Version Control tool feature in MorphX makes it possible to use a version control system, such as Microsoft Visual SourceSafe (VSS) or Visual Studio Team Foundation Server (TFS), to keep track of changes to elements in the AOT. The tool is accessible from several places: from the Version Control menu in the Development Workspace, from toolbars in the AOT and X++ code editor, and from the context menu on elements in the AOT.

Using a version control system offers several benefits:

- **Revision history of all elements** All changes are captured, along with a description of the change, making it possible to consult the change history and retrieve old versions of an element.
- **Code quality enforcement** The implementation of version control in Microsoft Dynamics AX enables a fully configurable quality standard for all check-ins. With the quality standard, all changes are verified according to coding practices. If a change doesn't meet the criteria, it is rejected.
- **Isolated development** Each developer can have a local installation and make all modifications locally. When modifications are ready, they can be checked in and made available to consumers of the build. A developer can rewrite fundamental areas of the system without causing instability issues for others. Developers are also unaffected by any downtime of a centralized development server.

Even though using a version control system is optional, it is strongly recommended that you consider one for any development project. Microsoft Dynamics AX 2012 supports three version control systems: VSS 6.0 and TFS, which are designed for large development projects, and MorphX VCS. MorphX VCS is designed for smaller development projects that previously couldn't justify the additional overhead that using a version control system server adds to the process. Table 2-8 shows a side-by-side comparison of the version control system options.

TABLE 2-8 Overview of version control systems.

	No version control system	MorphX VCS	VSS	TFS
Application Object Servers required	1	1	1 for each developer	1 for each developer
Database servers required	1	1	1 for each developer	1 for each developer
Build process required	No	No	Yes	Yes
Master file	Model store	Model store	XPOs	XPOs
Isolated development	No	No	Yes	Yes
Multiple checkout	N/A	No	Configurable	Configurable
Change description	No	Yes	Yes	Yes

	No version control system	MorphX VCS	VSS	TFS
Change history	No	Yes	Yes	Yes
Change list support (atomic check-in of a set of files)	N/A	No	No	Yes
Code quality enforcement	No	Configurable	Configurable	Configurable

The elements persisted on the version control server are file representations of the elements in the AOT. The file format used is the standard Microsoft Dynamics AX export format (.xpo). Each .xpo file contains only one root element.

There are no additional infrastructure requirements when you use MorphX VCS, which makes it a perfect fit for partners running many parallel projects. In such setups, each developer often works simultaneously on several projects, toggling between projects and returning to past projects. In these situations, the benefits of having a change history are enormous. With just a few clicks, you can enable MorphX VCS to persist the a changes in the business database. Although MorphX VCS provides many of the same capabilities as a version control server, it has some limitations. For example, MorphX VCS does not provide any tools for maintenance, such as making backups, archiving, or labeling.

In contrast, VSS and TFS are designed for large projects in which many developers work together on the same project for an extended period of time (for example, an independent software vendor building a vertical solution).

Figure 2-32 shows a typical deployment using VSS or TFS, in which each developer locally hosts the AOS and the database. Each developer also needs a copy of all .xpo files. When a developer communicates with the version control server, the .xpo files are transmitted.

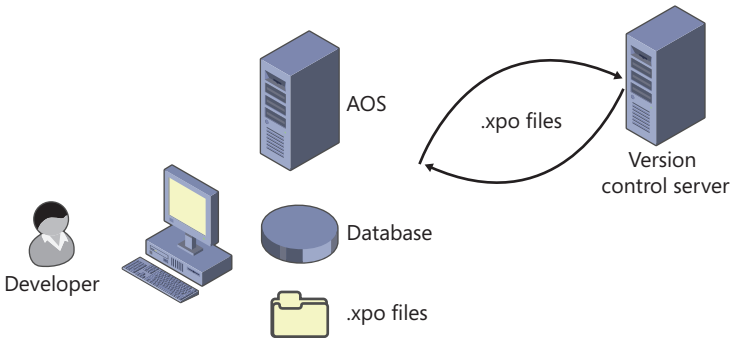


FIGURE 2-32 Typical deployment using version control.

Note In earlier versions of Microsoft Dynamics AX, a Team Server was required to assign unique IDs as elements were created. Microsoft Dynamics AX 2012 uses a new ID allocation scheme, which eliminates the need for the Team Server. For more information element IDs, see Chapter 21.

Element life cycle

Figure 2-33 shows the element life cycle in a version control system. When an element is in a state marked with a lighter shade, it can be edited; otherwise, it is read-only.

You can create an element in two ways:

- Create a new element.
- Customize an existing element, resulting in an *overlayered* version of the element. Because elements are stored for each layer in the version control system, customizing an element effectively creates a new element.

After you create an element, you must add it to the version control system. First, give it a proper name in accordance with naming conventions, and then click Add To Version Control on the context menu. After you create the element, you must check it in.

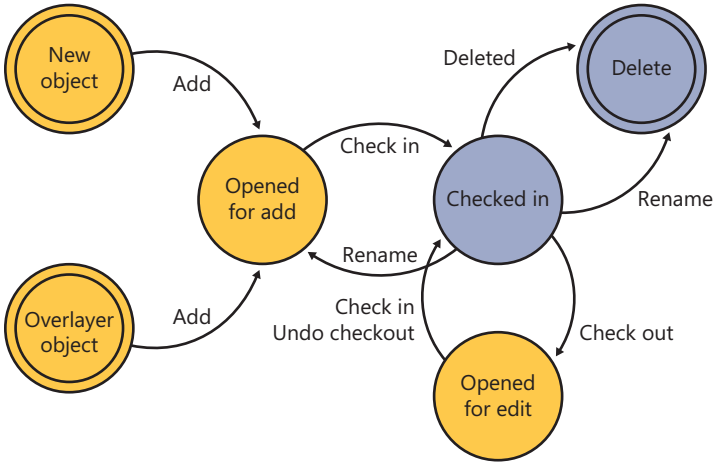


FIGURE 2-33 Element life cycle.

An element that is checked in can be renamed. Renaming an element deletes the element with the old name and adds an element with the new name.

Quality checks

Before the version control system accepts a check-in, it might subject the elements to quality checks. You define what is accepted in a check-in when you set up the version control system. The following checks are supported:

- Compiler errors
- Compiler warnings
- Compiler tasks
- Best practice errors

When a check is enabled, it is carried out when you do a check-in. If the check fails, the check-in stops. You must address the issue and restart the check-in.

Source code casing

You can set the Source Code Title Case Update tool, available on the Add-Ins submenu, to execute automatically before elements are checked in to ensure uniform casing in variable and parameter declarations and references. You can specify this parameter when setting up the version control system by selecting the Run Title Case Update check box.

Common version control tasks

Table 2-9 describes some of the tasks that are typically performed with a version control system. Later sections describe additional tasks that you can perform when using version control with Microsoft Dynamics AX.

TABLE 2-9 Version control tasks.

Action	Description
Check out an element	To modify an element, you must check it out. Checking out an element locks it so that others can't modify it while you're working. To see which elements you have currently checked out, on the Microsoft Dynamics AX menu, click Control > Pending Objects. The elements you've checked out (or that you've created and not yet checked in), appear in blue, rather than black, in the AOT.
Undo a checkout	If you decide that you don't want to modify an element that you checked out, you can undo the checkout. This releases your lock on the element and imports the most recent checked-in revision of the element to undo your changes.
Check in an element	When you have finalized your modifications, you must check in the elements for them to be part of the next build. When you click Check-In on the context menu, the dialog box shown in Figure 2-34 appears, displaying all the elements that you currently have checked out. The Check In dialog box shows all open elements by default; you can remove any elements not required in the check-in from the list by pressing Alt+F9.

	The following procedure is recommended for checking in your work: ■ Perform synchronization to update all elements in your environment to the latest version. ■ Verify that everything is still working as intended. Compilation is not enough. ■ Check in the elements.
Create an element	When using version control, you create new elements just as you normally would in the MorphX environment without a version control system. These elements are not part of your check-in until you click Add To Version Control on the context menu. You can also create all element types except those listed in System Settings (on the Development Workspace Version Control menu, point to Control > Setup > System Settings). By default, jobs and private projects are not accepted. New elements should follow Microsoft Dynamics AX naming conventions. The best practice naming conventions are enforced by default, so you can't check in elements with names such as *aaaElement*, *DEL_Element*, *element1*, or *element2*. (The only *DEL_ elements* allowed are those required for version upgrade purposes.) You can change naming requirements in System Settings.
Rename an element	An element must be checked in to be renamed. Because all references between .xpo files are strictly name-based, all references to renamed elements must be updated. For example, if you rename a table field, you must also update any form or report that uses that field. Most references in metadata in the AOT are ID-based and thus, they are not affected when an element is renamed; in most cases, it is enough to check out the form or report and include it in the check-in to update the .xpo file. You can use the cross-reference tool to identify references. References in X++ code are name-based. You can use the compiler to find affected references. An element's revision history is kept intact when elements are renamed. No tracking information in the version control system is lost because of an element is renamed.
Delete an element	You delete an element as you normally would in Microsoft Dynamics AX. The delete operation must be checked in before the deletion is visible to other users of the version control system. You can see pending deletions in the Pending Objects dialog box.
Get the latest version of an element	If someone else has checked in a new version of an element, you can use the Get Latest option on the context menu to get the version of the element that was checked in most recently. This option isn't available if you have the element checked out yourself. Get Latest is not available with MorphX VCS.

Figure 2-34 shows the Check In dialog box.

Work with labels

Working with labels is similar to working with elements. To change, delete, or add a label, you must check out the label file containing the label. You can check out the label file from the Label editor dialog box.

The main difference between checking out elements and checking out label files is that simultaneous checkouts are allowed for label files. This means that others can change labels while you have a label file checked out.

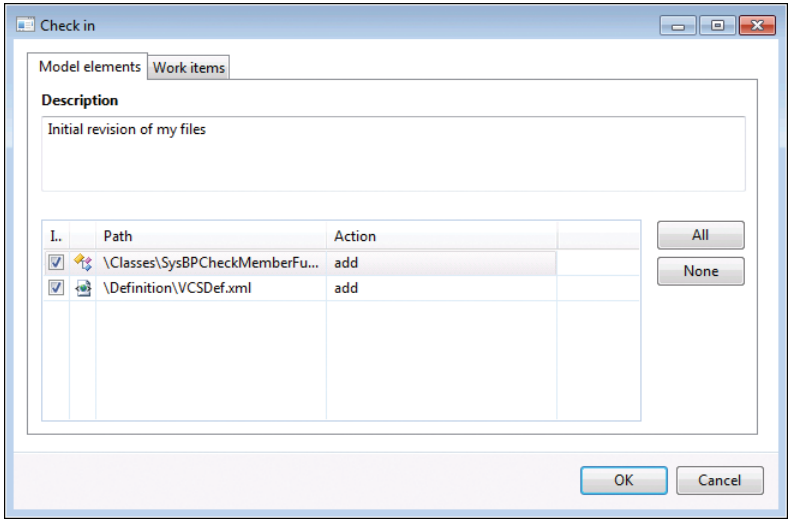


FIGURE 2-34 The Check In dialog box.

If you create a new label when using version control, a temporary label ID is assigned (for example, @$AA0007 as opposed to @USR1921). When you check in a label file, your changes are automatically merged into the latest version of the file and the temporary label IDs are updated. All references in the code are automatically updated to the newly assigned label IDs. Temporary IDs eliminate the need for a central Team Server, which was required for Microsoft Dynamics AX 2009, because IDs no longer have to be assigned when the labels are created. If you modify or delete a label that another person has also modified or deleted, your conflicting changes are abandoned. Such lost changes are shown in the Infolog after the check-in completes.

Synchronize elements

Synchronization makes it possible for you to get the latest version of all elements. This step is required before you can check in any elements. You can initiate synchronization from the Development Workspace. On the Version Control menu, point to Periodic > Synchronize.

Synchronization is divided into three operations that happen automatically in the following sequence:

1. Copy the latest files from the version control server to the local disk.
2. Import the files into the AOT.
3. Compile the imported files.

Use synchronization to make sure your system is up to date. Synchronization won't affect any new elements that you have created or any elements that you have checked out.

Figure 2-35 shows the Synchronization dialog box.

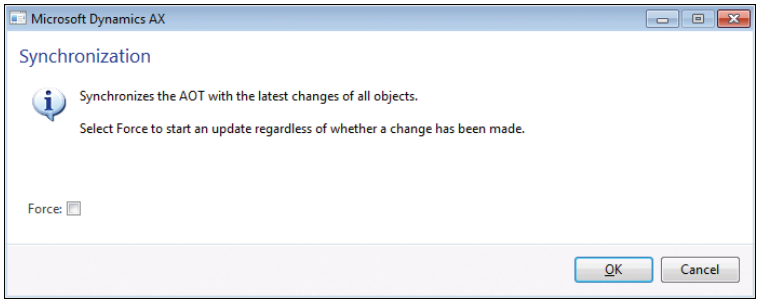


FIGURE 2-35 The Synchronization dialog box.

Selecting the Force check box gets the latest version of all files, even if they haven't changed, and then imports every file.

When using VSS, you can also synchronize to a label defined in VSS. This way, you can easily synchronize to a specific build or version number.

Synchronization is not available with MorphX VCS.

View the synchronization log

The way that you keep track of versions on the client depends on your version control system. VSS requires that Microsoft Dynamics AX keep track of itself. When you synchronize the latest version, it is copied to the local repository folder from the version control system. Each file must be imported into Microsoft Dynamics AX to be reflected in the AOT. To minimize the risk of partial synchronization, a log entry is created for each file. When all files are copied locally, the log is processed, and the files are automatically imported into Microsoft Dynamics AX.

When synchronization fails, the import operation is usually the cause of the problem. Synchronization failure leaves your system in a partially synchronized state. To complete the synchronization, restart Microsoft Dynamics AX and restart the import. You use the synchronization log to restart the import, and you access it from the Development Workspace menu at Version Control > Inquiries > Synchronization log.

The Synchronization Log dialog box, shown in Figure 2-36, displays each batch of files, and you can restart the import operation by clicking Process. If the Completed check box is not selected, the import has failed and should be restarted.

The Synchronization log is not available with MorphX VCS.

Show the history of an element

One of the biggest advantages of version control is the ability to track changes to elements. Selecting History on an element's context menu displays a list of all changes to an element, as shown in Figure 2-37.

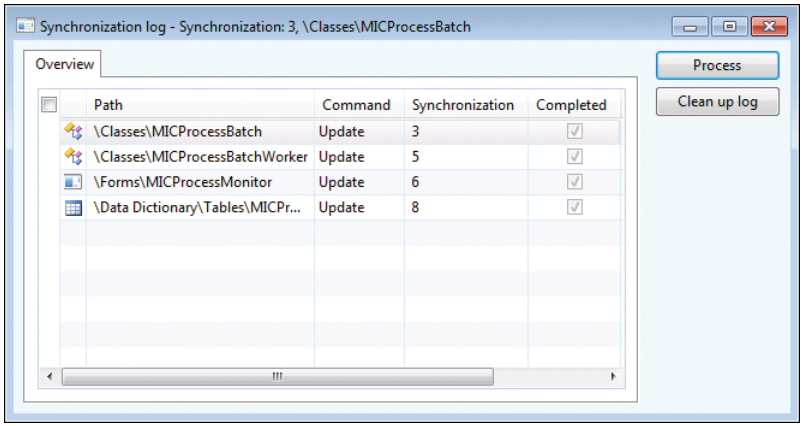


FIGURE 2-36 The Synchronization Log dialog box.

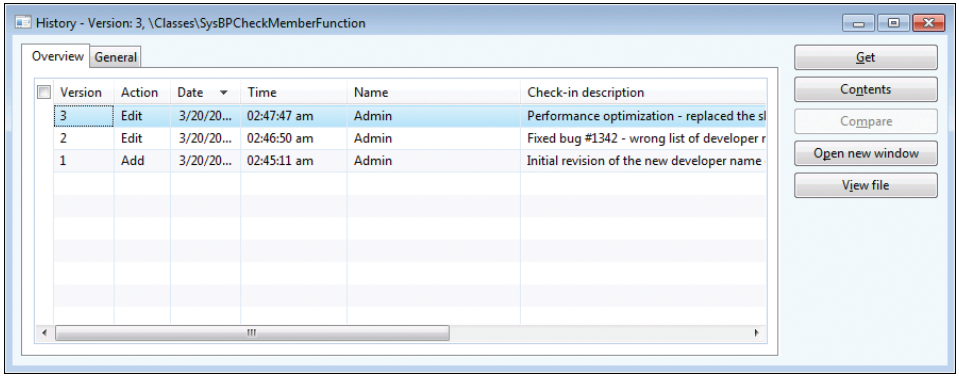


FIGURE 2-37 Revision history of an element.

This dialog box shows the version number, the action performed, the time the action was performed, and who performed the action. You can also see the change number and the change description.

A set of buttons in the History dialog box allows further investigation of each version. Clicking Contents opens a form that shows other elements included in the same change. Clicking Compare opens the Compare dialog box, where you can do a line-by-line comparison of two versions of the element. The Open New Window button opens an AOT window that shows the selected version of the element, which is useful for investigating properties because you can use the standard MorphX toolbox. Clicking View File opens the .xpo file for the selected version in Notepad.

Compare revisions

Comparison is the key to harvesting the benefits of a version control system. You can start a comparison from several places, including from the context menu of an element by pointing to Compare. Figure 2-38 shows the Comparison dialog box, where two revisions of the form *CustTable* are selected.

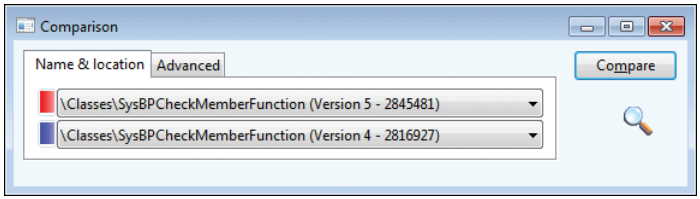


FIGURE 2-38 Comparing element revisions from version control.

The Compare dialog box contains a list of all checked-in versions, in addition to the element versions available in other layers installed.

View pending elements

When you're working on a project, it's easy to lose track of which elements you've opened for editing. The Pending Objects dialog box, shown in Figure 2-39, lists the elements that are currently checked out in the version control system. Notice the column containing the action performed on the element. Deleted elements are available only in this dialog box; they are no longer shown in the AOT.

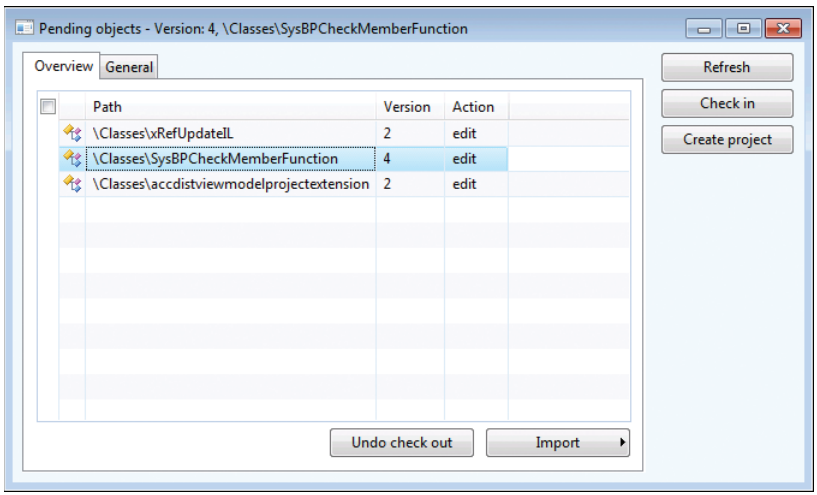


FIGURE 2-39 Pending elements.

You can access the Pending Objects dialog box from the Development Workspace menu: Version Control > Pending Objects.

Create a build

Because the version control system contains .xpo files and not a model file, a build process is required to generate the model file from the .xpo files. The following procedure provides a high-level overview of the build process:

1. Use the CombineXPOs command-line utility to combine all .xpo files into one. This step makes the .xpo file consumable by Microsoft Dynamics AX. Microsoft Dynamics AX requires all referenced elements to be present in the .xpo file or to already exist in the AOT to maintain the references during import.
2. Import the new .xpo file by using the command-line parameter -AOTIMPORTFILE=<*FileNam*e. xpo> -MODEL=<*Model Name*> to Ax32.exe. This step imports the .xpo file and compiles everything. After this step is complete, the new model is ready in the model store.
3. Export the model to a file by using the axutil command-line utility: axutil export / model:<*model name*> /file:<*model file name*>.
4. Follow these steps for each layer and each model that you build.

The build process doesn't apply to MorphX VCS.

Integrate Microsoft Dynamics AX with other version control systems

The implementation of the version control system in Microsoft Dynamics AX is fully pluggable. This means that any version control system can be integrated with Microsoft Dynamics AX.

Integrating with another version control system requires a new class implementing the *SysVersionControlFileBasedBackEnd* interface. It is the implementation's responsibility to provide the communication with the version control system server being used.

CHAPTER 3

Microsoft Dynamics AX and .NET

In this chapter

Introduction 73
Use third-party assemblies 74
Write managed code 77
Hot swap assemblies on the server 84

Introduction

Complex systems, such as Microsoft Dynamics AX 2012, are often deployed in heterogeneous environments that contain several disparate systems. Often, these systems contain legacy data that might be required for running Microsoft Dynamics AX, or they might offer functionality that is vital for running the organization.

Microsoft Dynamics AX 2012 offers several ways of integrating with other systems. For example, your organization might need to harvest information from old Microsoft Excel files. To do this, you could write a simple add-on in Microsoft Visual Studio and easily integrate it with Microsoft Dynamics AX. Or your organization might have a legacy system that is physically located in a distant location that requires invoice information to be sent to it in a fail-safe manner. In this case, you could set up a message queue to perform the transfers. You could use the Microsoft .NET Framework to interact with the message queue from within Microsoft Dynamics AX.

This chapter focuses on some of the ways that you can integrate Microsoft Dynamics AX with other systems by taking advantage of managed code through X++ code. One way is to consume managed code directly from X++ code; another way is to author or extend existing business logic in managed code by using the Visual Studio environment. To facilitate this interoperability, Microsoft Dynamics AX provides the managed code with managed classes (called *proxies*) that represent X++ artifacts. This allows you to write managed code that uses the functionality these proxies provide in a type-safe and convenient manner.

In both cases, the .NET Framework provides access to the functionality, and this functionality is used in Microsoft Dynamics AX.

Note You can also make Microsoft Dynamics AX functionality available to other systems by using services. For more information, see Chapter 12, "Microsoft Dynamics AX services and integration."

Use third-party assemblies

Sometimes, you can implement the functionality that you are looking to provide by using a managed component (a .NET assembly) that you purchase from a third-party vendor. Using these dynamic-link libraries (DLLs) can be—and often is—more cost effective than writing the code yourself. These components are wrapped in managed assemblies in the form of .dll files, along with their Program Database (PDB) files, which contain symbol information that is used in debugging, and their XML files, which contain documentation that is used for IntelliSense in Visual Studio. Typically, these assemblies come with an installation program that often installs the assemblies in the global assembly cache (GAC) on the computer that consumes the functionality. This computer can be either on the client tier, on the server tier, or both. Only assemblies with strong names can be installed in the GAC.

Use strong-named assemblies

It is always a good idea to use a DLL that has a strong name, which means that the DLL is signed by the author, regardless of whether the assembly is stored in the GAC. This is true for assemblies that are installed on both the client tier and the server tier. A strong name defines the assembly's identity by its simple text name, version number, and culture information (if provided)—plus a public key and a digital signature. Assemblies with the same strong name are expected to be identical.

Strong names satisfy the following requirements:

- **Guarantee name uniqueness by relying on unique key pairs.** No one can generate the same assembly name that you can, because an assembly generated with one private key has a different name than an assembly generated with another private key.
- **Protect the version lineage of an assembly.** A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that the version of the assembly that they are loading comes from the same publisher that created the version the application was built with.
- **Provide a strong integrity check.** Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. Note, however, that by themselves, strong names do not imply a level of trust such as that provided by a digital signature and supporting certificate.

When you reference a strong-named assembly, you can expect certain benefits, such as versioning and naming protection. If the strong-named assembly references an assembly with a simple name, which does not have these benefits, you lose the benefits that you derive by using a strong-named assembly and open the door to possible DLL conflicts. Therefore, strong-named assemblies can reference only other strong-named assemblies.

If the assembly that you are consuming does not have a strong name, and is therefore not installed in the GAC, you must manually copy the assembly (and the assemblies it depends on, if applicable) to a directory where the .NET Framework can find it when it needs to load the assembly for execution.

It is a good practice to place the assembly in the same directory as the executable that will ultimately load it (in other words, the folder on the client or the server in which the application is located). You might also want to store the assembly in the Client\Bin directory (even if it is used on the server exclusively), so that the client can pick it up and use it for IntelliSense.

Reference a managed DLL from Microsoft Dynamics AX

Microsoft Dynamics AX 2012 does not have a built-in mechanism for bulk deployment or installation of a particular DLL on client or server computers, because each third-party DLL has its own installation process. You must do this manually by using the installation script that the vendor provides or by placing the assemblies in the appropriate folders.

After you install the assembly on the client or server computer, you must add a reference to the assembly in Microsoft Dynamics AX, so that you can program against it in X++. You do this by adding the assembly to the *References* node in the Application Object Tree (AOT).

To do this, right-click the *References* node, and then click Add Reference. A dialog box like the one shown in Figure 3-1 appears.

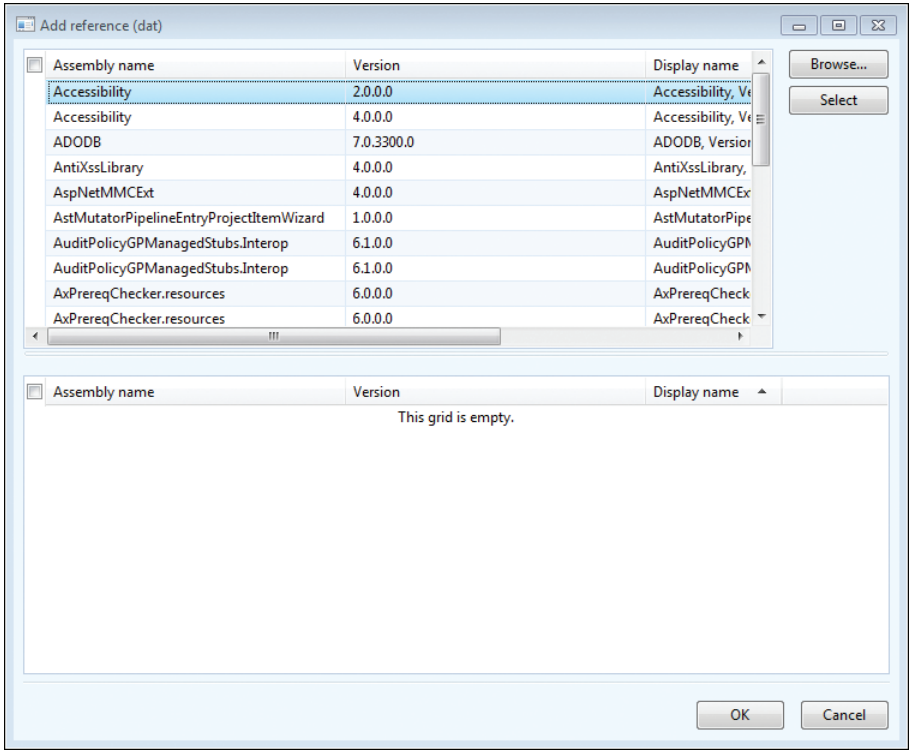


FIGURE 3-1 Adding a reference to a third-party assembly.

The top pane of the dialog box shows the assemblies that are installed in the GAC. If your assembly is installed in the GAC, click Select to add the reference to the *References* node. If the assembly is located in either the Client\Bin or the Server\Bin binary directory, click Browse. A file browser dialog box will appear where you can select your assembly. After you choose your assembly, it will appear in the bottom pane and will be added when you click OK.

Code against the assembly in X++

After you add the assembly, you are ready to use it from X++. If you install the code in the Client\Bin directory, IntelliSense features are available to help you edit the code. You can now use the managed code features of X++ to instantiate public managed classes, call methods on them, and so on. For more information, see Chapter 4, "The X++ programming language."

Note that there are some limitations to what you can achieve in X++ when calling managed code. One such limitation is that you cannot easily code against generic types (or execute generic methods). Another stems from the way the X++ interpreter works. Any managed object is represented as an instance of type *ClrObject*, and this has some surprising manifestations. For instance, consider the following code:

```
static void TestClr(Args _args)
{
    if (System.Int32::Parse("0"))
    {
        print "Do not expect to get here";
    }
    pause;
}
```

Obviously, you wouldn't expect the code in the *if* statement to execute because the result of the managed call is *0*, which is interpreted as *false*. However, the code actually prints the string literal because the return value of the call is a *ClrObject* instance that is not null (in other words, *true*). You can solve these problems by storing results in variables before use: the assignment operator will correctly unpack the value, as shown in the following example:

```
static void TestClr(Args _args)
{
    int i = System.Int32::Parse("0");
    if (i)
    {
        print "Do not expect to get here";
    }
    pause;
}
```

Write managed code

Sometimes your requirements cannot be satisfied by using an existing component and you have to roll up your sleeves and develop some code—in either C# or VB.NET. Microsoft Dynamics AX has great provisions for this: the integration features between Microsoft Dynamics AX and Visual Studio give you the luxury of dealing with X++ artifacts (classes, tables, and enumerations) as managed classes that behave the way that a developer of managed code would expect. The Microsoft Dynamics AX Business Connector manages the interaction between the two environments. Broadly speaking, you can create a project in Visual Studio as you normally would, and then add that project to the Visual Studio *Projects* node in the AOT. This section walks you through the process.

This example shows how to create managed code in C# (VB.NET could also be used) that reads the contents of an Excel spreadsheet and inserts the contents into a table in Microsoft Dynamics AX. This example is chosen to illustrate the concepts described in this chapter rather than for the functionality it provides.

The process is simple: you author the code in Visual Studio, and then add the solution to Application Explorer, which is just the name for the AOT in Visual Studio. Then, functionality from Microsoft Dynamics AX is made available for consumption by the C# code, which illustrates the proxy feature.

Assume that the Excel file contains the names of customers and the date that they registered as customers with your organization, as shown in Figure 3-2.

	A	B	C	D	E	F
1	Name	Date				
2	Abercrombie, Kim	04/28/74				
3	Abolrous, Hazem	06/22/49				
4	Abrus, Luka	03/19/47				
5	Abu-Dayah, Ahmad	08/11/65				
6	Acevedo, Humberto	04/17/85				
7	Achong, Gustavo	09/09/80				
8	Ackerman, Pilar	09/08/81				
9	Adalsteinsson, Gudmundur	04/17/86				
10	Adams, Terry	06/22/49				
11	Affronti, Michael					

Sheet1 Sheet2 Sheet3

FIGURE 3-2 Excel spreadsheet that contains a customer list.

Also assume that you've defined a table (called, say, CustomersFromExcel) in the AOT that will end up containing the information, subject to further processing. You could go about reading the information from the Excel files from X++ in several ways: one is by using the Excel automation model; another is by manipulating the Office Open XML document by using the XML classes. However, because it is so easy to read the contents of Excel files by using ADO.NET, that is what you decide to do. You start Visual Studio, create a C# class library called *ReadFromExcel*, and then write the following code:

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace Contoso
{
    using System.Data;
    using System.Data.OleDb;
    public class ExcelReader
    {
        static public void ReadDataFromExcel(string filename)
        {
            string connectionString;
            OleDbDataAdapter adapter;
            connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;"
            + "Data Source=" + filename + ";"
            + "Extended Properties='Excel 12.0 Xml;"

            + "HDR=YES'"; // Since sheet has row with column titles

            adapter = new OleDbDataAdapter(
                "SELECT * FROM [sheet1$]",
                connectionString);
            DataSet ds = new DataSet();
            // Get the data from the spreadsheet:
            adapter.Fill(ds, "Customers");
            DataTable table = ds.Tables["Customers"];
            foreach (DataRow row in table.Rows)
            {
                string name = row["Name"] as string;
                DateTime d = (DateTime)row["Date"];
            }
        }
    }
}
```

The *ReadDataFromExcel* method reads the data from the Excel file given as a parameter, but it does not currently do anything with that data. You still need to establish a connection to the Microsoft Dynamics AX system to store the values in the table. There are several ways of doing this, but in this case, you will simply use the Microsoft Dynamics AX table from the C# code by using the proxy feature.

The first step is to make the Visual Studio project (that contains the code) into a Microsoft Dynamics AX "citizen." You do this by selecting the Add ReadFromExcel To AOT menu item on the Visual Studio project. When this is done, the project is stored in the AOT and can use all of the functionality that is available for nodes in the AOT. The project can be stored in separate layers, be imported and exported, and so on. The project is stored in its entirety, and you can open Visual Studio to edit the project by clicking Edit on the context menu, as shown in Figure 3-3.

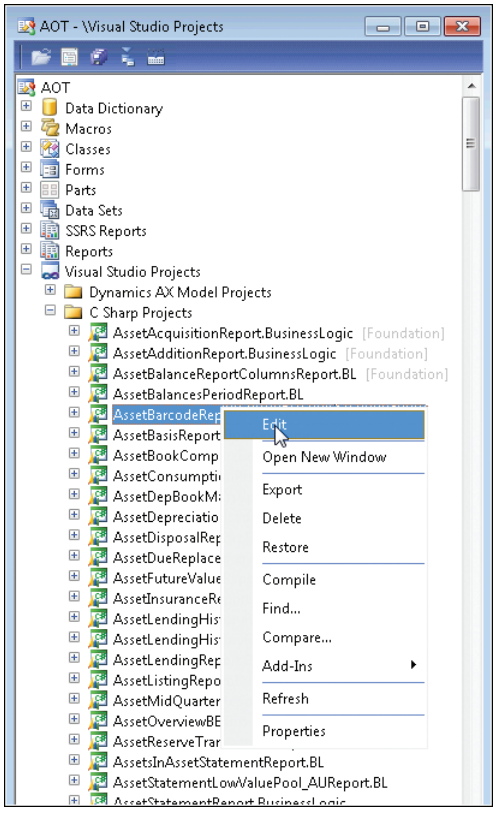


FIGURE 3-3 Context menu for Visual Studio projects that are stored in the AOT.

Tip You can tell that a project has been added to the AOT because the Visual Studio project icon is updated with a small Microsoft Dynamics AX icon in the lower-left corner.

With that step out of the way, you can use the version of the AOT that is available in Application Explorer in Visual Studio to fetch the table to use in the C# code (see Figure 3-4). If the Application Explorer window is not already open, you can open it by clicking Application Explorer on the View menu.

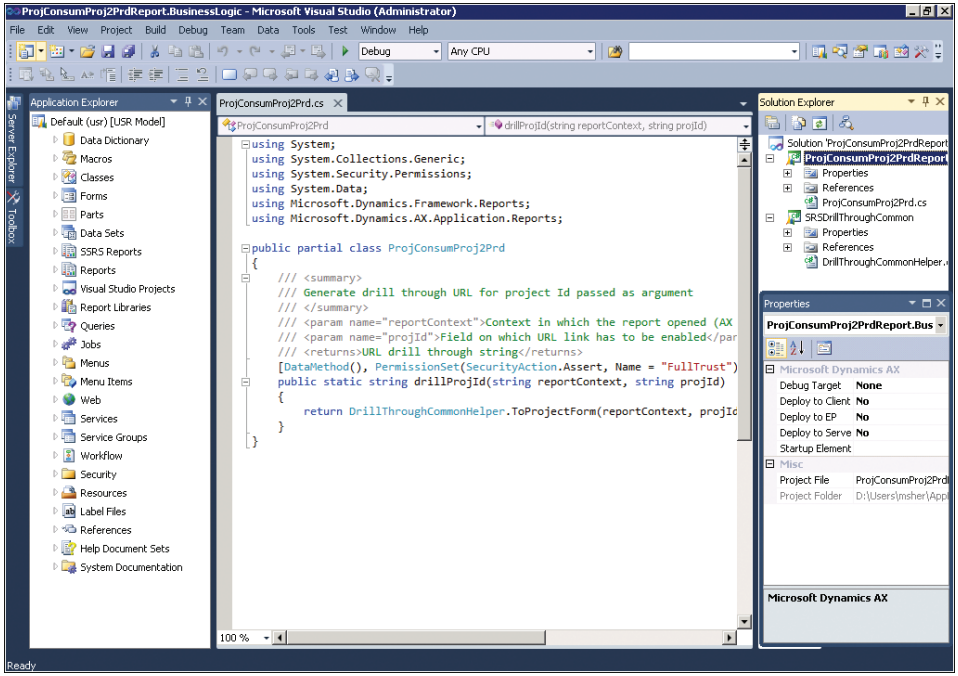


FIGURE 3-4 Application Explorer with a Microsoft Dynamics AX project open.

You can then create a C# representation of the table by dragging the table node from Application Explorer into the project.

After you drag the table node into the Visual Studio project, you will find an entry in the project that represents the table. The items that you drag into the project in this way are now available to code against in C#, just as though they had been written in C#. This happens because the drag operation creates a proxy for the table under the covers; this proxy takes care of the plumbing required to communicate with the Microsoft Dynamics AX system, while presenting a high-fidelity managed interface to the developer.

You can now proceed by putting the missing pieces into the C# code to write the data into the table. Modify the code as shown in the following example:

```
DataTable table = ds.Tables["Customers"];
var customers = new ReadFromExcel.CustomersFromExcel();
foreach (DataRow row in table.Rows)
{
    string name = row["Name"] as string;
    DateTime d = (DateTime)row["Date"];
    customers.Name = name;
    customers.Date = d;
    customers.Write();
}
```

Note The table from Microsoft Dynamics AX is represented just like any other type in C#. It supports IntelliSense, and the documentation comments that were added to methods in X++ are available to guide you as you edit.

The data will be inserted into the CustomersFromExcel table as it is read from the ADO.NET table that represents the contents of the spreadsheet. However, before either the client or the server can use this code, you must deploy it. You can do this by setting the properties in the Properties window for the Microsoft Dynamics AX project in Visual Studio. In this case, the code will run on the client, so you set the *Deploy to Client* property to *Yes*. There is a catch, though: you cannot deploy the assembly to the client when the client is running, so you must close any Microsoft Dynamics AX clients prior to deployment.

To deploy the code, right-click the Visual Studio project, and then click Deploy. If all goes well, a *Deploy Succeeded* message will appear in the status line.

Note You do not have to add a reference to the assembly because a reference is added implicitly to projects that you add to the AOT. You only need to add references to assemblies that are not the product of a project that has been added to the AOT.

As soon as you deploy the assembly, you can code against it in X++. The following example illustrates a simple snippet in an X++ job:

```
static void ReadCustomers(Args _args)
{
    ttsBegin;
    Contoso.ExcelReader::ReadDataFromExcel(@"c:\Test\customers.xlsx");
    ttsCommit;
}
```

When this job runs, it calls into the managed code and insert the records into the Microsoft Dynamics AX database.

Debug managed code

To ease the process of deploying after building, Visual Studio properties let you define what happens when you run the Microsoft Dynamics AX project. You manage this by using the *Debug Target* and *Startup Element* properties. You can enter the name of an element to execute—typically, a class with a suitable main method or a job. When you start the project in Visual Studio, it will create a new instance of the client and execute the class or job. The X++ code then calls back into the C# code where breakpoints are set. For more information, see "Debugging Managed Code in Microsoft Dynamics AX" at *http://msdn.microsoft.com/en-us/library/gg889265.aspx*.

An alternative to using this feature is to attach the Visual Studio debugger to the running Microsoft Dynamics AX client (by using the Attach To Process menu item on the Debug menu in Visual Studio). You can then set breakpoints and use all of the functionality of the debugger that you normally would. If you are running the Application Object Server (AOS) on your own computer, you can attach to that as well, but you must have administrator privileges to do so.

Important Do not debug in a production environment.

Proxies

As you can see, wiring up managed code to work with Microsoft Dynamics AX is quite simple because of the proxies that are generated behind the scenes to represent the Microsoft Dynamics AX tables, enumerations, and classes. In developer situations, it is standard to develop the artifacts in Microsoft Dynamics AX iteratively and then code against them in C#. This process is seamless because the proxies are regenerated by Visual Studio at build time, so that they are always synchronized with the corresponding artifacts in the AOT; in other words, the proxies never become out of date. In this way, proxies for Microsoft Dynamics AX artifacts differ from Visual Studio proxies for web services. These proxies are expected to have a stable application programming interface (API) so that the server hosting the web service is not contacted every time the project is built. Proxies are generated not only for the items that the user has chosen to drop onto the *Project* node as described previously. For instance, when a proxy is generated for a class, proxies will also be generated for all of its base classes, along with all artifacts that are part of the parameters for any methods, and so on.

To see what the proxies look like, place the cursor on a given proxy name in the code editor, such as *CustomersFromExcel* in the example, right-click, and then click Go To Definition (or use the convenient keyboard shortcut F12). All of the proxies are stored in the Obj/Debug folder for the project. If you look carefully, you will notice that the proxies use the Microsoft Dynamics AX Business Connector to do the work of interfacing with the Microsoft Dynamics AX system. The Business Connector has been completely rewritten from the previous version to support this scenario; in older versions of the product, the Business Connector invariably created a new session through which the interaction occurred. This is not the case for the new version of the Business Connector (at least when it is used as demonstrated here). That is why the transaction that was started in the job shown earlier is active when the records are inserted into the table. In fact, all aspects of the user's session are available to the managed code. This is the crucial difference between authoring business logic in managed code and consuming the business logic from managed code. When you author business logic, the managed code becomes an extension to the X++ code, which means that you can crisscross between Microsoft Dynamics AX and managed code in a consistent environment. When consuming business logic, you are better off using the services framework that Microsoft Dynamics AX provides, and then consuming the service from your application. This has big benefits in terms of scalability and deployment flexibility.

Figure 3-5 shows how the Business Connector relates to Microsoft Dynamics AX and .NET application code.

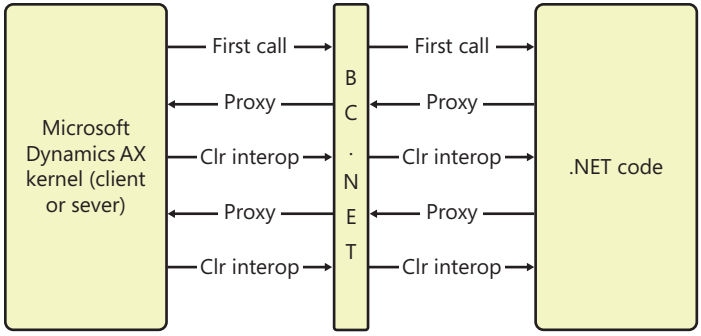


FIGURE 3-5 Interoperability between Microsoft Dynamics AX and .NET code through the Business Connector.

To demonstrate the new role of the Business Connector, the following example opens a form in the client that called the code:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace OpenFormInClient
{
    public class OpenFormClass
    {
        public void DoOpenForm(string formName)
        {
            Args a = new Args();
            a.name = formName;
            var fr = new FormRun(a);
            fr.run();
            fr.detach();
        }
    }
}
```

In the following example, a job is used to call managed code to open the CustTable form:

```
static void OpenFormFromDotNet(Args _args)
{
    OpenFormInClient.OpenFormClass opener;
    opener = new OpenFormInClient.OpenFormClass();
    opener.DoOpenForm("CustTable");
}
```

Note The *FormRun* class in this example is a kernel class. Because only an application class is represented in Application Explorer, you cannot add this proxy by dragging and dropping as described earlier. Instead, drop any class from Application Explorer onto the Visual Studio project, and then set the file name property of the class to *Class.*<kernelclassname>*.axproxy*. In this example, the name would be *Class.FormRun.axproxy*.

This would not have been possible with older versions of the Business Connector because they were basically faceless clients that could not display any user interface. Now, the Business Connector is actually part of the client (or server), and therefore, it can do anything they can. In Microsoft Dynamics AX 2012 R2, you can still use the Business Connector as a stand-alone client, but that is not recommended because that functionality is now better implemented by using services (see Chapter 12). The Business Connector that is included with Microsoft Dynamics AX is built with .NET Framework 3.5. That means that it is easier to build the business logic with this version of .NET; if you cannot do that for some reason, you must add markup to the App.config file to compensate. If you are using a program that is running .NET Framework 4.0 and you need to use the Business Connector through the proxies as described, you would typically add the following markup to the App.config file for your application:

```
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    "<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

Hot swap assemblies on the server

The previous section described how to express business logic in managed code. To simplify the scenario, code running on the client was used as an example. This section describes managed code running on the server.

You designate managed code to run on the server by setting the *Deploy to Server* property for the project to Yes, as shown in Figure 3-6.

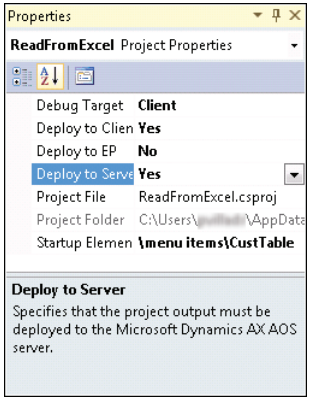


FIGURE 3-6 Property sheet showing the *Deploy to Server* property set to Yes.

When you set this property as shown in Figure 3-6, the assembly is deployed to the server directory. If the server has been running for a while, it will typically have loaded the assemblies into the current application domain. If Visual Studio were to deploy a new version of an existing assembly, the deployment would fail because the assembly would already be loaded into the current application domain.

To avoid this situation, the server has the option to start a new application domain in which it executes code from the new assembly. When a new client connects to the server, it will execute the updated code in a new application domain, while already connected clients continue to use the old version.

To use the hot-swapping feature, you must enable the option in the Microsoft Dynamics AX Server Configuration Utility by selecting the Allow Hot Swapping of Assemblies When The Server Is Running check box, as shown in Figure 3-7. To open the Microsoft Dynamics AX Server Configuration Utility, on the Start menu, point to Administrative Tools, and then click Microsoft Dynamics AX 2012 Server Configuration.

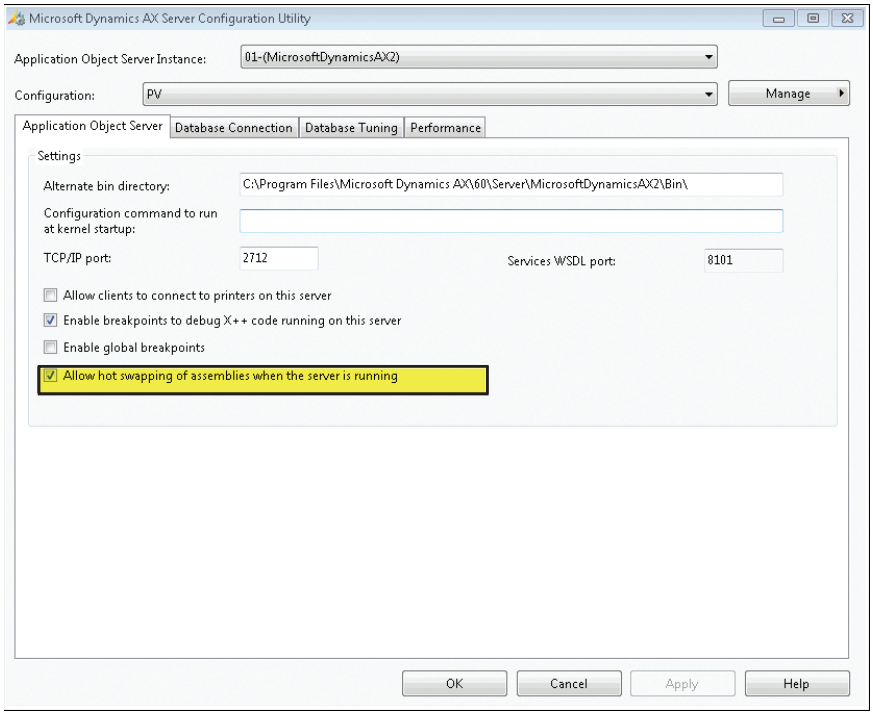


FIGURE 3-7 Allow hot swapping by using the Microsoft Dynamics AX Server Configuration Utility.

Note The example in the previous section illustrated how to run and debug managed code on the client, which is safe because the code runs only on a development computer. You can debug code that is running on the server (by starting Visual Studio as a privileged user and attaching to the server process as described in the "Debug managed code" section). However, you should never do this on a production server because any breakpoints that are encountered will stop all of the managed code from running, essentially blocking any users who are logged on to the server and processes that are running. Another reason you should not use hot swapping in a production scenario is that calling into another application domain extracts a performance overhead. The feature is intended only for development scenarios, where the performance of the application is irrelevant.

CHAPTER 4

The X++ programming language

In this chapter

Introduction 87
Jobs 88
The type system 88
Syntax 93
Classes and interfaces 117
Code access security 124
Compiling and running X++ as .NET CIL 126
Design and implementation patterns 128

Introduction

X++ is an object-oriented, application-aware, and data-aware programming language. The language is object oriented because it supports object abstractions, abstraction hierarchies, polymorphism, and encapsulation. It is application aware because it includes keywords such as *client*, *server*, *changecompany*, and *display* that are useful for writing client/server enterprise resource planning (ERP) applications. And it is data aware because it includes keywords such as *firstFast*, *forceSelectOrder*, and *forUpdate*, as well as a database query syntax, that are useful for programming database applications.

You use the Microsoft Dynamics AX designers and tools to edit the structure of application types. You specify the behavior of application types by writing X++ source code in the X++ editor. The X++ compiler compiles this source code into bytecode intermediate format. Model data, X++ source code, intermediate bytecode, and .NET common intermediate language (CIL) code are stored in the model store.

The Microsoft Dynamics AX runtime dynamically composes object types by loading overridden bytecode from the highest-level definition in the model layering stack. Objects are instantiated from these dynamic types. Similarly, the compiler produces .NET CIL from the X++ source code from the highest layer. For more information about the Microsoft Dynamics AX layering technology, see Chapter 21, "Application models."

This chapter describes the Microsoft Dynamics AX runtime type system and the features of the X++ language that are essential to writing ERP applications. It will also help you avoid common programming pitfalls that stem from implementing X++. For an in-depth discussion of the type system and the X++ language, refer to the Microsoft Dynamics AX 2012 software development kit (SDK), available on MSDN.

Jobs

Jobs are globally defined functions that execute in the Windows client run-time environment. Developers frequently use jobs to test a piece of business logic because they are easily executed from within the MorphX development environment, by either pressing F5 or selecting Go on the command menu. However, you shouldn't use jobs as part of your application's core design. The examples provided in this chapter can be run as jobs.

Jobs are model elements that you create by using the Application Object Tree (AOT). The following X++ code provides an example of a job model element that prints the string "Hello World" to an automatically generated window. The *pause* statement stops program execution and waits for user input from a dialog box.

```
static void myJob(Args _args)
{
    print "Hello World";
    pause;
}
```

The type system

The Microsoft Dynamics AX runtime manages the storage of value type data on the call stack and reference type objects on the memory heap. The *call stack* is the memory structure that holds data about the active methods called during program execution. The *memory heap* is the memory area that allocates storage for objects that are destroyed automatically by the Microsoft Dynamics AX runtime.

Value types

Value types include the built-in primitive types, extended data types, enumeration types, and built-in collection types:

- The primitive types are *boolean*, *int*, *int64*, *real*, *date*, *utcDateTime*, *timeofday*, *str*, and *guid*.
- The extended data types are specialized primitive types and specialized base enumerations.
- The enumeration types are base enumerations and extended data types.
- The collection types are the built-in array and container types.

By default, variables declared as value types are assigned their zero value by the Microsoft Dynamics AX runtime. These variables can't be set to null. Variable values are copied when variables are used to invoke methods and when they are used in assignment statements. Therefore, two value type variables can't reference the same value.

Reference types

Reference types include the record types, class types, and interface types:

- The record types are *table*, *map*, and *view*. User-defined record types are dynamically composed from application model layers. Microsoft Dynamics AX runtime record types are exposed in the system application programming interface (API).

Note Although the methods are not visible in the AOT, all record types implement the methods that are members of the system *xRecord* type, a Microsoft Dynamics AX runtime class type.

- User-defined class types are dynamically composed from application model layers and Microsoft Dynamics AX runtime class types exposed in the system API.
- Interface types are type specifications and can't be instantiated in the Microsoft Dynamics AX runtime. Class types can, however, implement interfaces.

Variables declared as reference types contain references to objects that the Microsoft Dynamics AX runtime instantiates from dynamically composed types defined in the application model layering system and from types exposed in the system API. The Microsoft Dynamics AX runtime also performs memory deallocation (garbage collection) for these objects when they are no longer referenced. Reference variables declared as record types reference objects that the Microsoft Dynamics AX runtime instantiates automatically. Class type objects are programmatically instantiated using the *new* operator. Copies of object references are passed as reference parameters in method calls and are assigned to reference variables, so two variables can reference the same object.

More Info Not all nodes in the AOT name a type declaration. Some class declarations are merely *syntactic sugar*—convenient, human-readable expressions. For example, the class header definition for all rich client forms declares a *FormRun* class type. *FormRun* is also, however, a class type in the system API. Allowing this declaration is syntactic sugar because it is technically impossible for two types to have the same name in the Microsoft Dynamics AX class type hierarchy.

Type hierarchies

The X++ language supports the definition of type hierarchies that specify generalized and specialized relationships between class types and table types. For example, a check payment method is a type of payment method. A type hierarchy allows code reuse. Reusable code is defined on base types defined higher in a type hierarchy because they are inherited, or reused, by derived types defined lower in a type hierarchy.

Tip You can use the Type Hierarchy Context and Type Hierarchy Browser tools in MorphX to visualize, browse, and search the hierarchy of any type.

The following sections introduce the base types provided by the Microsoft Dynamics AX runtime and describe how they are extended in type hierarchies.

Caution The Microsoft Dynamics AX type system is known as a weak type system because X++ accepts certain type assignments that are clearly erroneous and lead to runtime errors. Be aware of the caveats outlined in the following sections, and try to avoid weak type constructs when writing X++ code.

The *anytype* type

The Microsoft Dynamics AX type system doesn't have a single base type from which all types ultimately derive. However, the *anytype* type imitates a base type for all types. Variables of the *anytype* type function like value types when they are assigned a value type variable and like reference types when they are assigned a reference type variable. You can use the *SysAnyType* class to explicitly box all types, including value types, and make them function like reference types.

The *anytype* type, shown in the following code sample, is syntactic sugar that allows methods to accept any type as a parameter or allows a method to return different types:

```
static str queryRange(anytype _from, anytype _to)
{
    return SysQuery::range(_from,_to);
}
```

You can declare variables by using *anytype*. However, the underlying data type of an *anytype* variable is set to match the first assignment, and you can't change its type afterward, as shown here:

```
anytype a = 1;
print strfmt("%1 = %2", typeof(a), a); //Integer = 1
a = "text";
print strfmt("%1 = %2", typeof(a), a); //Integer = 0
```

The *common* type

The *common* type is the base type of all record types. Like the *anytype* type, record types are context-dependent types whose variables can be used as though they reference single records or as a record cursor that can iterate over a set of database records.

By using the *common* type, you can cast one record type to another (possibly incompatible) record type, as shown in this example:

```
//customer = vendor; //Compile error
common = customer;
vendor = common;     //Accepted
```

Tables in Microsoft Dynamics AX also support inheritance and polymorphism. This capability offers a type-safe method of sharing commonalities such as methods and fields between tables. It is possible to override table methods but not table fields. A base table can be marked as abstract or final through the table's properties.

Table maps defined in the AOT are a type-safe method of capturing commonalities between record types across type hierarchies, and you should use them to prevent incompatible record assignments. A table map defines fields and methods that safely operate on one or more record types.

The compiler doesn't validate method calls on the *common* type. For example, the compiler accepts the following method invocation, even though the method doesn't exist:

```
common.nonExistingMethod();
```

For this reason, you should use reflection to confirm that the method on the *common* type exists before you invoke it, as shown in this example. For more information, see Chapter 20, "Reflection."

```
if (tableHasMethod(new DictTable(common.tableId), identifierStr(existingMethod)))
{
    common.existingMethod();
}
```

The *object* type

The built-in *object* type is a weak reference type whose variables reference objects that are instances of class or interface types in the Microsoft Dynamics AX class hierarchy.

The type system allows you to implicitly cast base type objects to derived type objects and to cast derived type objects to base type objects, as shown here:

```
baseClass = derivedClass;
derivedClass = baseClass;
```

The *object* type allows you to use the assignment operator and cast one class type to another, incompatible class type, as shown in the following code. The probable result of this action, however, is a run-time exception when your code encounters an object of an unexpected type.

```
Object myObject;
//myBinaryIO = myTextIO; //Compile error
myObject = myTextIO;
mybinaryIO = myObject;   //Accepted
```

Use the *is* and *as* operators instead of the assignment operator to prevent these incompatible type casts. The *is* operator determines if an instance is of a particular type, and the *as* operator casts an instance as a particular type, or null if they are not compatible. The *is* and *as* operators work on class and table types.

```
myTextIO = myObject as TextIO;
if (myBinaryIO is TextIO)
{
}
```

You can use the *object* type for late binding to methods, similar to the dynamic keyword in C#. Keep in mind that a run-time error will occur if the method invoked doesn't exist.

```
myObject.lateBoundMethod();
```

Extended data types

You use the AOT to create extended data types that model concrete data values and data hierarchies. For example, the *Name* extended data type is a *string*, and the *CustName* and *VendName* extended data types extend the *Name* data type.

The X++ language supports extended data types but doesn't offer type checking according to the hierarchy of extended data types. X++ treats any extended data type as its primitive type; therefore, code such as the following is allowed:

```
CustName customerName;
FileName fileName = customerName;
```

When used properly, extended data types improve the readability of X++ code. It's easier to understand the intended use of a *CustName* data type than a *string* data type, even if both are used to declare *string* variables.

Extended data types are more than just type definitions that make X++ code more readable. On each extended data type, you can also specify how the system displays values of this type to users. Further, you can specify a reference to a table. The reference enables the form's rendering engine to automatically build lookup forms for form controls by using the extended data type, even when the form controls are not bound to a data source. On string-based extended data types, you can specify the maximum string size of the type. The database layer uses the string size to define the underlying columns for fields that use the extended data type. Defining the string size in only one place makes it easy to change.

Syntax

The X++ language belongs to the "curly brace" family of programming languages (those that use curly braces to delimit syntax blocks), such as C, C++, C#, and Java. If you're familiar with any of these languages, you won't have a problem reading and understanding X++ syntax.

Unlike many programming languages, X++ is not case sensitive. However, using camel casing *(camelCasing)* for variable names and Pascal casing *(PascalCasing)* for type names is considered a best practice. (More best practices for writing X++ code are available in the Microsoft Dynamics AX 2012 SDK.) You can use the Source Code Titlecase Update tool (accessed from the Add-Ins submenu in the AOT) to automatically apply casing in X++ code to match the best practice recommendation.

Common language runtime (CLR) types, which are case sensitive, are one important exception to the casing guidelines. For information about how to use CLR types, see the "CLR interoperability" section later in this chapter.

Variable declarations

You must place variable declarations at the beginning of methods. Table 4-1 provides examples of value type and reference type variable declarations, in addition to example variable initializations. Parameter declaration examples are provided in the "Classes and interfaces" section later in this chapter.

TABLE 4-1 X++ variable declaration examples.

Type	Examples
anytype	`anytype type = null;` `anytype type = 1;`
Base enumeration types	`NoYes theAnswer = NoYes::Yes;`
boolean	`boolean b = true;`

container	`container c1 = ["a string", 123];`
	`container c2 = [["a string", 123], c1];`
	`container c3 = connull();`

Type	Examples
date	`date d = 31\12\2008;`
Extended data types	`Name name = "name";`
guid	`guid g = newguid();`
int	`int i = -5;` `int h = 0xAB;`

int64	`int64 i = -5;`
	`int64 h = 0xAB;`
	`int64 u = 0xA0000000u;`

Object types	`Object obj = null;`
	`MyClass myClass = new MyClass();`
	`System.Text.StringBuilder sb = new System.Text.StringBuilder();`

real	`real r1 = 3.14;` `real r2 = 1.0e3;`
Record types	`Common myRecord = null;` `CustTable custTable = null;`

str	`str s1 = "a string";`
	`str s2 = 'a string';`
	`str 40 s40 = "string 40";`

TimeOfDay	`TimeOfDay time = 43200;`
utcDateTime	`utcDateTime dt = 2008-12-31T23:59:59;`

Note String literals can be expressed using either single or double quotes. It is considered best practice to use single quotes for system strings, like file names, and double quotes for user interface strings. The examples in this chapter adhere to this guideline.

Declaring variables with the same name as their type is a best practice. At first glance, this approach might seem confusing. Consider this class and its getter/setter method to its field:

```
Class Person
{
    Name name;

    public Name Name(Name _name = name)
    {
        name = _name;
        return name;
    }
}
```

Because X++ is not case sensitive, the word *name* is used in eight places in the preceding code. Three refer to the extended data type, four refer to the field, and one refers to the method (*_name* is used twice). To improve readability, you could rename the variable to something more specific,

such as *personName*. However, using a more specific variable name implies that a more specific type should be used (and created if it doesn't already exist). Changing both the type name and the variable name to *PersonName* wouldn't improve readability. The benefit of this practice is that if you know the name of a variable, you also know its type.

Note Previous versions of Microsoft Dynamics AX required a dangling semicolon to signify the end of a variable declaration. This is no longer required because the compiler solves the ambiguity by reading one token ahead, except where the first statement is a static CLR call. The compiler still accepts the now-superfluous semicolons, but you can remove them if you want to.

Expressions

X++ expressions are sequences of operators, operands, values, and variables that yield a result. Table 4-2 summarizes the types of expressions allowed in X++ and includes examples of their use.

TABLE 4-2 X++ expression examples.

Category	Examples

Access operators	`this //Instance member access`
	`element //Form member access`
	`<datasource>_ds //Form data source access`
	`<datasource>_q //Form query access`
	`x.y //Instance member access`
	`E::e //Enum access`
	`a[x] //Array access`
	`[v1, v2] = c //Container access`
	`Table.Field //Table field access`
	`Table.(FieldId) //Table field access`
	`(select statement).Field //Select result access`
	`System.Type //CLR namespace type access`
	`System.DayOfWeek::Monday //CLR enum access`

Arithmetic operators	`x = y + z // Addition`
	`x = y - z // Subtraction`
	`x = y * z // Multiplication`
	`x = y / z // Division`
	`x = y div z // Integer division`
	`x = y mod z // Integer division remainder`

Bitwise operators	`x = y & z // Bitwise AND`
	`x = y | z // Bitwise OR`
	`x = y ^ z // Bitwise exclusive OR (XOR)`
	`x = ~z // Bitwise complement`

Conditional operators	`x ? y : z`

Logical operators	`if (!obj) // Logical NOT`
	`if (a && b) // Logical AND`
	`if (a || b) // Logical OR`

Method invocations	`super() //Base member invocation`
	`MyClass::m() //Static member invocation`
	`myObject.m() //Instance member invocation`
	`this.m() //This instance member invocation`
	`myTable.MyMap::m(); //Map instance member invocation`
	`f() //Built-in function call`

Category	Examples

Object creation operators	`new MyClass()  //X++ object creation`
	`new System.DateTime()  //CLR object wrapper and`
	`//CLR object creation`
	`new System.Int32[100]() //CLR array creation`

Parentheses	`(x)`

Relational operators	`x < y  // Less than`
	`x > y  // Greater than`
	`x <= y  // Less than or equal`
	`x >= y  // Greater than or equal`
	`x == y  // Equal`
	`x != y  // Not equal`
	`select t where t.f like "a*"  // Select using wildcards`

Shift operators	`x = y << z  // Shift left` `x = y >> z  // Shift right`
String concatenation	`"Hello" + "World"`
Values and variables	`"string"` `myVariable`

Statements

X++ statements specify object state and object behavior. Table 4-3 provides examples of X++ language statements that are commonly found in many programming languages. In-depth descriptions of each statement are beyond the scope of this book.

TABLE 4-3 X++ statement examples.

Statement	Example

.NET CLR interoperability statement	`System.Text.StringBuilder sb;`
	`sb = new System.Text.StringBuilder();`
	`sb.Append("Hello World");`
	`print sb.ToString();`
	`pause;`

Assignment statement	`int i = 42;`
	`i = 1;`
	`i++;`
	`++i;`
	`i--;`
	`--i;`
	`i += 1;`
	`i -= 1;`
	`this.myDelegate += eventhandler(obj.handler);`
	`this.myDelegate -= eventhandler(obj.handler);`

break statement	`int i;`
	`for ( i = 0; i < 100; i++ )`
	`{`
	`  if ( i > 50 )`
	`  {`
	`  break;`
	`  }`
	`}`

breakpoint statement	`breakpoint; //Causes the debugger to be invoked`

Statement	Example
Casting statement	`MyObject myObject = object as MyObject;` `boolean isCompatible = object is MyObject;`

changeCompany statement	`MyTable myTable;`
	`while select myTable`
	`{`
	`  print myTable.myField;`
	`}`
	`changeCompany("ZZZ")`
	`{`
	`  while select myTable`
	`  {`
	`  print myTable.myField;`
	`  }`
	`}`
	`pause;`

Compound statement	`int i;`
	`{`
	`  i = 3;`
	`  i++;`
	`}`

continue statement	`int i;`
	`int j = 0;`
	`for( i = 0; i < 100; i++ )`
	`{`
	`  if ( i < 50 )`
	`  {`
	`  continue;`
	`  }`
	`  j++;`
	`}`

do while statement	`int i = 4;`
	`do`
	`{`
	`  i++;`
	`}`
	`while ( i <= 100 );`

flush statement	`MyTable myTable;` `flush myTable;`

for statement	`int i;`
	`for ( i = 0; i < 42; i++ )`
	`{`
	`  print i;`
	`}`
	`pause;`

if statement	`boolean b = true;`
	`int i = 42;`
	`if ( b == true )`
	`{`
	`  i++;`
	`}`
	`else`
	`{`
	`  i--;`
	`}`

Statement	Example

Local function	`static void myJob(Args _args)`
	`{`
	`  str myLocalFunction()`
	`  {`
	`  return "Hello World";`
	`  }`
	`  print myLocalFunction();`
	`  pause;`
	`}`

pause statement	`print "Hello World";` `pause;`

print statement	`int i = 42;`
	`print i;`
	`print "Hello World";`
	`print "Hello World" at 10,5;`
	`print 5.2;`
	`pause;`

retry statement	`try`
	`{`
	`  throw error("Force exception");`
	`}`
	`catch( exception::Error )`
	`{`
	`  retry;`
	`}`

return statement	`int foo()`
	`{`
	`  return 42;`
	`}`

switch statement	`str s = "test";`
	`switch ( s )`
	`{`
	`  case "test" :`
	`  print s;`
	`  break;`
	`  default :`
	`  print "fail";`
	`}`
	`pause;`

System function	`guid g = newGuid();` `print abs(-1);`
throw statement	`throw error("Error text");`

try statement	`try`
	`{`
	`  throw error("Force exception");`
	`}`
	`catch( exception::Error )`
	`{`
	`  print "Error";`
	`  pause;`
	`}`
	`catch`
	`{`
	`  print "Another exception";`
	`  pause;`
	`}`

unchecked statement	`unchecked(Uncheck::TableSecurityPermission)`
	`{`
	`  this.method();`
	`}`

Statement	Example

while statement	`int i = 4;`
	`while ( i <= 100 )`
	`{`
	`  i++;`
	`}`

window statement	`window 100, 10 at 100, 10;`
	`print "Hello World";`
	`pause;`

Data-aware statements

The X++ language has built-in support for querying and manipulating database data. The syntax for database statements is similar to Structured Query Language (SQL), and this section assumes that you're familiar with SQL. The following code shows how a *select* statement is used to return only the first selected record from the MyTable database table and how the data in the record's *myField* field is printed:

```
static void myJob(Args _args)
{
     MyTable myTable;
     select firstOnly * from myTable where myTable.myField1 == "value";
     print myTable.myField2;
     pause;
}
```

The "* from" part of the *select* statement in the example is optional. You can replace the asterisk (*) character with a comma-separated field list, such as *myField2, myField3*. You must define all fields, however, on the selection table model element, and only one selection table is allowed mmediately after the *from* keyword. The *where* expression in the *select* statement can include any number of logical and relational operators. The *firstOnly* keyword is optional and can be replaced by one or more of the optional keywords. Table 4-4 describes all possible keywords. For more information about database-related keywords, see Chapter 17, "The database layer."

TABLE 4-4 Keyword options for *select* statements.

Keyword	Description
crossCompany	Forces the Microsoft Dynamics AX runtime to generate a query without automatically adding the *where* clause in the *dataAreaId* field. This keyword can be used to select records from all or from a set of specified company accounts. For example, the query `while select crosscompany:companies myTable { }` selects all records in the *myTable* table from the company accounts specified in the *companies* container.
firstFast	Fetches the first selected record faster than the remaining selected records.

Keyword	Description
firstOnly *firstOnly1*	Returns only the first selected record.
firstOnly10	Returns only the first 10 selected records.
firstOnly100	Returns only the first 100 selected records.
firstOnly1000	Returns only the first 1,000 selected records.
forceLiterals	Forces the Microsoft Dynamics AX runtime to generate a query with the specified field constraints. For example, the query generated for the preceding code example looks like this: *select * from myTable where myField1='value'*. Database query plans aren't reused when this option is specified. This keyword can't be used with the *forcePlaceholders* keyword.
forceNestedLoop	Forces the SQL Server query processor to use a nested-loop algorithm for table join operations. Other join algorithms, such as hash-join and merge-join, are therefore not considered by the query processor.
forcePlaceholders	Forces the Microsoft Dynamics AX runtime to generate a query with placeholder field constraints. For example, the query generated for the preceding code example looks like this: *select * from myTable where myField1=?*. Database query plans are reused when this option is specified. This is the default option for *select* statements that don't join table records. This keyword can't be used with the *forceLiterals* keyword.
forceSelectOrder	Forces the Microsoft SQL Server query processor to access tables in the order in which they are specified in the query.
forUpdate	Selects records for updating.
generateOnly	Instructs the SQL Server query processor to only generate the SQL statements—and not execute them. The generated SQL statement can be retrieved using the *getSQLStatement* method on the primary table.
noFetch	Specifies that the Microsoft Dynamics AX runtime should not execute the statement immediately because the records are required only by some other operation.
optimisticLock	Overrides the table's *OccEnabled* property and forces the optimistic locking scheme. This keyword can't be used with the *pessimisticLock* and *repeatableRead* keywords.
pessimisticLock	Overrides the table's *OccEnabled* property and forces the pessimistic locking scheme. This keyword can't be used with the *optimisticLock* and *repeatableRead* keywords.
repeatableRead	Locks all records read within a transaction. This keyword can be used to ensure consistent data is fetched by identical queries for the duration of the transaction, at the cost of blocking other updates of those records. Phantom reads can still occur if another process inserts records that match the range of the query. This keyword can't be used with the *optimisticLock* and *pessimisticLock* keywords.
reverse	Returns records in the reverse of the *select* order.
validTimeState	Instructs the SQL Server query processor to use the provided date or date range instead of the current date. For example, the query `while select validTimeState(fromDate, toDate) myTable { }` selects all records in the *myTable* table that are valid in the period from *fromDate to toDate*.

The following code example demonstrates how to use a table index clause to suggest the index that a database server should use when querying tables. The Microsoft Dynamics AX runtime appends an *order by* clause and the *index* fields to the first *select* statement's database query. Records are thus ordered by the index. The Microsoft Dynamics AX runtime can insert a query hint into the second *select* statement's database query, if the hint is feasible to use.

```
static void myJob(Args _args)
{
    MyTable1 myTable1;
    MyTable2 myTable2;

    while select myTable1
        index myIndex1
    {
        print myTable1.myField2;
    }

    while select myTable2
        index hint myIndex2
    {
        print myTable2.myField2;
    }
    pause;
}
```

The following code example demonstrates how the results from a *select* query can be ordered and grouped. The first *select* statement specifies that the resulting records must be sorted in ascending order based on *myField1* values and then in descending order based on *myField2* values. The second *select* statement specifies that the resulting records must be grouped by *myField1* values and then sorted in descending order.

```
static void myJob(Args _args)
{
    MyTable myTable;

    while select myTable
        order by Field1 asc, Field2 desc
    {
        print myTable.myField;
    }
    while select myTable
        group by Field1 desc
    {
        print myTable.Field1;
    }
    pause;
}
```

The following code demonstrates use of the *avg* and *count* aggregate functions in *select* statements. The first *select* statement averages the values in the *myField* column and assigns the result to the *myField* field. The second *select* statement counts the number of records the selection returns and assigns the result to the *myField* field.

```
static void myJob(Args _args)
{
     MyTable myTable;

     select avg(myField) from myTable;
     print myTable.myField;

     select count(myField) from myTable;
     print myTable.myField;
     pause;
}
```

Caution The compiler doesn't verify that aggregate function parameter types are numeric, so the result that the function returns could be assigned to a field of type *string*. The result will be truncated if, for example, the *average* function calculates a value of 1.5 and the type of *myField* is an integer.

Table 4-5 Describes the aggregate functions supported in X++ *select* statements.

TABLE 4-5 Aggregate functions in X++ *select* statements.

Function	Description
avg	Returns the average of the non-null field values in the records the selection returns.
count	Returns the number of non-null field values in the records the selection returns.
maxOf	Returns the maximum of the non-null field values in the records the selection returns.
minOf	Returns the minimum of the non-null field values in the records the selection returns.
sum	Returns the sum of the non-null field values in the records the selection returns.

The following code example demonstrates how tables are joined with *join* conditions. The first *select* statement joins two tables by using an equality *join* condition between fields in the tables. The second *select* statement joins three tables to illustrate how you can nest *join* conditions and use an *exists* operator as an existence test with a *join* condition. The second *select* statement also demonstrates how you can use a *group by* sort in *join* conditions. In fact, the *join* condition can comprise multiple nested *join* conditions because the syntax of the *join* condition is the same as the body of a *select* statement.

```
static void myJob(Args _args)
{
     MyTable1 myTable1;
     MyTable2 myTable2;


     MyTable3 myTable3;

     select myField from myTable1
        join myTable2
           where myTable1.myField1=myTable2.myField1;
     print myTable1.myField;

     select myField from myTable1
        join myTable2
          group by myTable2.myField1
          where myTable1.myField1=myTable2.myField1
             exists join myTable3
                where myTable1.myField1=myTable3.mField2;
     print myTable1.myField;
     pause;
}
```

Table 4-6 describes the *exists* operator and the other *join* operators that can be used in place of the *exists* operator in the preceding example.

TABLE 4-6 Join operators.

Operator	Description
exists	Returns *true* if any records are in the result set after executing the *join* clause. Returns *false* otherwise.
notExists	Returns *false* if any records are in the result set after executing the *join* clause. Returns *true* otherwise.
outer	Returns the left outer *join* of the first and second tables.

The following example demonstrates use of the *while select* statement that increments the *myTable* variable's record cursor on each loop:

```
static void myJob(Args _args)
{
     MyTable myTable;

     while select myTable
     {
         Print myTable.myField;
     }
}
```

You must use the *ttsBegin*, *ttsCommit*, and *ttsAbort* transaction statements to modify records in tables and to insert records into tables. The *ttsBegin* statement marks the beginning of a database transaction block; *ttsBegin*-*ttsCommit* transaction blocks can be nested. The *ttsBegin* statements increment the transaction level; the *ttsCommit* statements decrement the transaction level. The outermost block decrements the transaction level to zero and commits all database inserts and updates performed since the first *ttsBegin* statement to the database. The *ttsAbort* statement rolls back all database inserts, updates, and deletions performed since the *ttsBegin* statement. Table 4-7 provides examples of these transaction statements for single records and operations and for set-based (multiple-record) operations.

The last example in Table 4-7 demonstrates the method *RowCount*. Its purpose is to get the count of records that are affected by set-based operations; namely, *insert_recordset*, *update_recordset*, and *delete_from*.

By using *RowCount,* it is possible to save one round-trip to the database in certain application scenarios; for example, when implementing insert-or-update logic.

TABLE 4-7 Transaction statement examples.

Statement Type	Example

delete_from	`MyTable myTable;`
	`Int64 numberOfRecordsAffected;`
	
	`ttsBegin;`
	`delete_from myTable`
	`  where myTable.id == "001";`
	`numberOfRecordsAffected = myTable.RowCount();`
	`ttsCommit;`

insert method	`MyTable myTable;`
	
	`ttsBegin;`
	`myTable.id = "new id";`
	`myTable.myField = "new value";`
	`myTable.insert();`
	`ttsCommit;`

insert_recordset	`MyTable1 myTable1;`
	`MyTable2 myTable2;`
	`int64 numberOfRecordsAffected;`
	
	`ttsBegin;`
	`insert_recordset myTable2 ( myField1, myField2 )`
	`  select myField1, myField2 from myTable1;`
	`numberOfRecordsAffected = myTable.RowCount();`
	`ttsCommit;`

select forUpdate	`MyTable myTable;`
	
	`ttsBegin;`
	`select forUpdate myTable;`
	`myTable.myField = "new value";`
	`myTable.update();`
	`ttsCommit;`

ttsBegin
ttsCommit
ttsAbort
	`boolean b = true;`
	
	`ttsBegin;`
	`if ( b == true )`
	`  ttsCommit;`
	`else`
	`  ttsAbort;`

Statement Type	Example

update_recordset	`MyTable myTable;`
	`int64 numberOfRecordsAffected;`
	
	`ttsBegin;`
	`update_recordset myTable setting`
	`  myField1 = "value1",`
	`  myField2 = "value2"`
	`  where myTable.id == "001";`
	`numberOfRecordsAffected = myTable.RowCount();`
	`ttsCommit;`

Exception handling

It is a best practice to use the X++ exception handling framework instead of programmatically halting a transaction by using the *ttsAbort* statement. An exception (other than the update conflict and duplicate key exceptions) thrown inside a transaction block halts execution of the block, and all of the inserts and updates performed since the first *ttsBegin* statement are rolled back. Throwing an exception has the additional advantage of providing a way to recover object state and maintain the consistency of database transactions. Inside the *catch* block, you can use the *retry* statement to run the *try* block again. The following example demonstrates throwing an exception inside a database transaction block:

```
static void myJob(Args _args)
{
    MyTable myTable;
    boolean state = false;

    try
    {
        ttsBegin;

        update_recordset myTable setting
            myField = "value"
            where myTable.id == "001";
        if(state==false)
        {
            throw error("Error text");
        }
        ttsCommit;
    }
    catch(Exception::Error)
    {
        state = true;
        retry;
    }
}
```

The *throw* statement throws an exception that causes the database transaction to halt and roll back. Code execution can't continue inside the scope of the transaction, so the runtime ignores

try and *catch* statements when inside a transaction. This means that an exception thrown inside a transaction can be caught only outside the transaction, as shown here:

```
static void myJob(Args _args)
{
    try
    {
        ttsBegin;
        try
        {
            ...
            throw error("Error text");
        }
        catch  //Will never catch anything
        {
        }
        ttsCommit;
    }
    catch(Exception::Error)
    {
        print "Got it";
        pause;
    }
    catch
    {
        print "Unhandled Exception";
        pause;
    }
}
```

Although a *throw* statement takes the exception enumeration as a parameter, using the *error* method to throw errors is considered best practice. The *try* statement's catch list can contain more than one *catch* block. The first *catch* block in the example catches error exceptions. The *retry* statement jumps to the first statement in the outer *try* block. The second *catch* block catches all exceptions not caught by *catch* blocks earlier in the *try* statement's catch list. Table 4-8 describes the Microsoft Dynamics AX system *Exception* data type enumerations that can be used in *try-catch* statements.

TABLE 4-8 *Exception* data type enumerations.

Element	Description
Break	Thrown when a user presses the Break key or Ctrl+C.
CLRError	Thrown when an unrecoverable error occurs in a CLR process.
CodeAccessSecurity	Thrown when an unrecoverable error occurs in the *demand* method of a *CodeAccessPermission* object.
DDEerror	Thrown when an error occurs in the use of a Dynamic Data Exchange (DDE) system class.
Deadlock	Thrown when a database transaction has deadlocked.

Element	Description
DuplicateKeyException	Thrown when a duplicate key error occurs during an insert operation. The *catch* block should change the value of the primary keys and use a *retry* statement to attempt to commit the halted transaction.
DuplicateKeyExceptionNotRecovered	Thrown when an unrecoverable duplicate key error occurs during an insert operation. The *catch* block shouldn't use a *retry* statement to attempt to commit the halted transaction.
*Error**	Thrown when an unrecoverable application error occurs. A *catch* block should assume that all database transactions in a transaction block have been halted and rolled back.
Internal	Thrown when an unrecoverable internal error occurs.
Numeric	Thrown when a mathematical error occurs like division by zero, logarithm of a negative number, or conversion between incompatible types.
PassClrObjectAcrossTiers	Thrown when an attempt is made to pass a CLR object from the client to the server tier or vice versa. The Microsoft Dynamics AX runtime doesn't support automatic marshaling of CLR objects across tiers.
Sequence	Thrown by the Microsoft Dynamics AX kernel if a database error or database operation error occurs.
Timeout	Thrown when a database operation times out.
UpdateConflict	Thrown when an update conflict error occurs in a transaction block using optimistic concurrency control. The *catch* block should use a *retry* statement to attempt to commit the halted transaction.
UpdateConflictNotRecovered	Thrown when an unrecoverable error occurs in a transaction block using optimistic concurrency control. The *catch* block shouldn't use a *retry* statement to attempt to commit the halted transaction.

* The *error* method is a static method of the global X++ class for which the X++ compiler allows an abbreviated syntax. The expression *Global::error("Error text")* is equivalent to the error expression in the code examples earlier in this section. Don't confuse these global X++ methods with Microsoft Dynamics AX system API methods, such as *newGuid*.

UpdateConflict and *DuplicateKeyException* are the only data exceptions that a Microsoft Dynamics AX application can handle inside a transaction. Specifically, with *DuplicateKeyException*, the database transaction isn't rolled back, and the application is given a chance to recover. *DuplicateKeyException* facilitates application scenarios (such as Master Planning) that perform batch processing and handles duplicate key exceptions without aborting the transaction in the midst of the resource-intensive processing operation.

The following example illustrates the usage of *DuplicateKeyException*:

```
static void DuplicateKeyExceptionExample(Args _args)
{
    MyTable myTable;

    ttsBegin;
    myTable.Name = "Microsoft Dynamics AX";
    myTable.insert();
    ttsCommit;

    ttsBegin;
    try
    {
```

```
            myTable.Name = "Microsoft Dynamics AX";
            myTable.insert();
        }
        catch(Exception::DuplicateKeyException)
        {
            info(strfmt("Transaction level: %1", appl.ttsLevel()));
            info(strfmt("%1 already exists.", myTable.Name));
            info(strfmt("Continuing insertion of other records"));
        }
        ttsCommit;
    }
```

In the preceding example, the *catch* block handles the duplicate key exception. Notice that the transaction level is still 1, indicating that the transaction hasn't aborted and the application can continue processing other records.

Note The special syntax where a table instance was included in the *catch* block is no longer available.

Interoperability

The X++ language includes statements that allow interoperability (interop) with .NET CLR assemblies and COM components. The Microsoft Dynamics AX runtime achieves this interoperability by providing Dynamics AX object wrappers around external objects and by dispatching method calls from the Microsoft Dynamics AX object to the wrapped object.

CLR interoperability

You can write X++ statements for CLR interoperability by using one of two methods: strong typing or weak typing. Strong typing is recommended because it is type safe and less error prone than weak typing, and it results in code that is easier to read. The MorphX X++ editor also provides IntelliSense as you type.

The examples in this section use the .NET *System.Xml* assembly, which is added as an AOT references node. (See Chapter 1, "Architectural overview," for a description of programming model elements.) The programs are somewhat verbose because the compiler doesn't support method invocations on CLR return types and because CLR types must be identified by their fully qualified name. For example, the expression *System.Xml.XmlDocument* is the fully qualified type name for the .NET Framework XML document type.

Caution X++ is case sensitive when referring to CLR types.

The following example demonstrates strongly typed CLR interoperability with implicit type conversions from Microsoft Dynamics AX strings to CLR strings in the string assignment statements and shows how CLR exceptions are caught in X++:

```
static void myJob(Args _args)
{
   System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
   System.Xml.XmlElement rootElement;
   System.Xml.XmlElement headElement;
   System.Xml.XmlElement docElement;
   System.String xml;
   System.String docStr = 'Document';
   System.String headStr = 'Head';
   System.Exception ex;
   str errorMessage;

   try
   {
       rootElement = doc.CreateElement(docStr);
       doc.AppendChild(rootElement);
       headElement = doc.CreateElement(headStr);
       docElement = doc.get_DocumentElement();
       docElement.AppendChild(headElement);
       xml = doc.get_OuterXml();
       print ClrInterop::getAnyTypeForObject(xml);
       pause;
   }
   catch(Exception::CLRError)
   {
       ex = ClrInterop::getLastException();
       if( ex )
       {
           errorMessage = ex.get_Message();
           info( errorMessage );
       }
   }
}
```

The following example illustrates how static CLR methods are invoked by using the X++ static method *accessor* *::*.

```
static void myJob(Args _args)
{
    System.Guid g = System.Guid::NewGuid();
}
```

The following example illustrates the support for CLR arrays:

```
static void myJob(Args _args)
{
    System.Int32 [] myArray = new System.Int32[100]();

    myArray.SetValue(1000, 0);
    print myArray.GetValue(0);
}
```

X++ supports passing parameters by reference to CLR methods. Changes that the called method makes to the parameter also change the caller variable's value. When non-object type variables are passed by reference, they are wrapped temporarily in an object. This operation is often called *boxing* and is illustrated in the following example:

```
static void myJob(Args _args)
{
    int myVar = 5;

    MyNamespace.MyMath::Increment(byref myVar);

    print myVar;   // prints 6
}
```

The called method could be implemented in C# like this:

```
// Notice: This example is C# code
static public void Increment(ref int value)
{
    value++;
}
```

Note Passing parameters by reference is supported only for CLR methods, not for X++ methods.

The second method of writing X++ statements for CLR uses weak typing. The following example shows CLR types that perform the same steps as in the first CLR interoperability example. In this case, however, all references are validated at run time, and all type conversions are explicit.

```
static void myJob(Args _args)
{
   ClrObject doc = new ClrObject('System.Xml.XmlDocument');
   ClrObject docStr;
```

```
    ClrObject rootElement;
    ClrObject headElement;
    ClrObject docElement;
    ClrObject xml;

    docStr = ClrInterop::getObjectForAnyType('Document');
    rootElement = doc.CreateElement(docStr);
    doc.AppendChild(rootElement);
    headElement = doc.CreateElement('Head');
    docElement = doc.get_DocumentElement();
    docElement.AppendChild(headElement);
    xml = doc.get_OuterXml();
    print ClrInterop::getAnyTypeForObject(xml);
    pause;
}
```

The first statement in the preceding example demonstrates the use of a static method to convert X++ primitive types to CLR objects. The *print* statement shows the reverse, converting CLR value types to X++ primitive types. Table 4-9 lists the value type conversions that Microsoft Dynamics AX supports.

TABLE 4-9 Type conversions supported in Microsoft Dynamics AX.

CLR Type	Microsoft Dynamics AX Type
Byte, SByte, Int16, UInt16, Int32	*int*
Byte, SByte, Int16, UInt16, Int32, Uint32, Int64	*int64*
DateTime	*utcDateTime*
Double, Single	*real*
Guid	*guid*
String	*str*
Microsoft Dynamics AX Type	**CLR Type**
int	*Int32, Int64*
int64	*Int64*
utcDateTime	*DateTime*
real	*Single, Double*
guid	*Guid*
str	*String*

The preceding code example also demonstrates the X++ method syntax used to access CLR object properties, such as *get_DocumentElement*. The CLR supports several operators that are not supported in X++. Table 4-10 lists the supported CLR operators and the alternative method syntax.

TABLE 4-10 CLR operators and methods.

CLR Operators	CLR Methods
Property operators	*get_<property>, set_<property>*
Index operators	*get_Item, set_Item*
Math operators	*op_<operation>(arguments)*

The following features of CLR can't be used with X++:

- Public fields (can be accessed by using CLR reflection classes)
- Events and delegates
- Generics
- Inner types
- Namespace declarations

COM interoperability

The following code example demonstrates COM interoperability with the XML document type in the Microsoft XML Core Services (MSXML) 6.0 COM component. The example assumes that you've installed MSXML. The MSXML document is first instantiated and wrapped in a Microsoft Dynamics AX COM object wrapper. A COM variant wrapper is created for a COM string. The direction of the variant is put into the COM component. The root element and head element variables are declared as COM objects. The example shows how to fill a string variant with an X++ string and then use the variant as an argument to a COM method, *loadXml*. The statement that creates the head element demonstrates how the Microsoft Dynamics AX runtime automatically converts Microsoft Dynamics AX primitive objects into COM variants.

```
static void Job2(Args _args)
{
    COM doc = new COM('Msxml2.DomDocument.6.0');
    COMVariant rootXml =
               new COMVariant(COMVariantInOut::In,COMVariantType::VT_BSTR);
    COM rootElement;
    COM headElement;

    rootXml.bStr('<Root></Root>');
    doc.loadXml(rootXml);
    rootElement = doc.documentElement();
    headElement = doc.createElement('Head');
    rootElement.appendChild(headElement);
    print doc.xml();
    pause;
}
```

Macros

With the macro capabilities in X++, you can define and use constants and perform conditional compilation. Macros are unstructured because they are not defined in the X++ syntax. Macros are handled before the source code is compiled. You can add macros anywhere you write source code: in methods and in class declarations.

Table 4-11 shows the supported macro directives.

TABLE 4-11 Macro directives.

Directive	Description
#define *#globaldefine*	Defines a macro with a value. `#define.MyMacro(SomeValue)` Defines the macro *MyMacro* with the value *SomeValue*.

#macro
...
#endmacro
 	
#localmacro
...
#endmacro
	Defines a macro with a value spanning multiple lines.
	`#macro.MyMacro`
	`  print "foo";`
	`  print "bar";`
	`#endmacro`
	Defines the macro *MyMacro* with a multiple-line value.

#macrolib	Includes a macro library. As a shorthand form of this directive, you can omit *macrolib*.
	`#macrolib.MyMacroLibrary`
	`#MyMacroLibrary`
	Both include the macro library *MyMacroLibrary*, which is defined under the Macros node in the AOT.

#MyMacro	Replaces a macro with its value.
	`#define.MyMacro("Hello World")`
	`print #MyMacro;`
	Defines the macro *MyMacro* and prints its value. In this example, "Hello World" would be printed.

#definc *#defdec*	Increments and decrements the value of a macro; typically used when the value is an integer. `#defdec.MyIntMacro` Decrements the value of the macro *MyIntMacro*.
#undef	Removes the definition of a macro. `#undef.MyMacro` Removes the definition of the macro *MyMacro*.

#if
...
#endif
	Conditional compile. If the macro referenced by the *#if* directive is defined or has a specific value, the following text is included in the compilation:
	`#if.MyMacro`
	`print "MyMacro is defined";`
	`#endif`

Directive	Description

	If *MyMacro* is defined, the *print* statement is included as part of the source code:
	`#if.MyMacro(SomeValue)`
	`print "MyMacro is defined and has value: SomeValue";`
	`#endif`
	
	If *MyMacro* has *SomeValue*, the *print* statement is included as part of the source code.

#ifnot
...
#endif
	Conditional compile. If the macro referenced by the *#ifnot* directive isn't defined or doesn't have a specific value, the following text is included in the compilation:
	`#ifnot.MyMacro`
	`print "MyMacro is not defined";`
	`#endif`
	
	If *MyMacro* is not defined, the *print* statement is included as part of the source code:
	`#ifnot.MyMacro(SomeValue)`
	`print "MyMacro does not have value: SomeValue; or it is not defined";`
	`#endif`
	
	If *MyMacro* is not defined, or if it does not have *SomeValue*, the *print* statement is included as part of the source code.

The following example shows a macro definition and reference:

```
void myMethod()
{
    #define.HelloWorld("Hello World")

    print #HelloWorld;
    pause;
}
```

As noted in Table 4-11, a macro library is created under the Macros node in the AOT. The library is included in a class declaration header or class method, as shown in the following example:

```
class myClass
{
    #MyMacroLibrary1
}
public void myMethod()
{
    #MyMacroLibrary2


    #MacroFromMyMacroLibrary1
    #MacroFromMyMacroLibrary2
}
```

A macro can also use parameters. The compiler inserts the parameters at the positions of the placeholders. The following example shows a local macro using parameters:

```
void myMethod()
{
    #localmacro.add
        %1 + %2
    #endmacro

    print #add(1, 2);
    print #add("Hello", "World");
    pause;
}
```

When a macro library is included or a macro is defined in the class declaration of a class, the macro can be used in the class and in all classes derived from the class. A subclass can redefine the macro.

Comments

X++ allows single-line and multiple-line comments. Single-line comments start with *//* and end at the end of the line. Multiple-line comments start with */** and end with ***/*. You can't nest multiple-line comments.

You can add reminders to yourself in comments that the compiler picks up and presents to you as tasks in its output window. To set up these tasks, start a comment with the word **TODO**. Be aware that tasks not occurring at the start of the comment, (for example, tasks that are deep inside multiple-line comments,) are ignored by the compiler.

The following code example contains comments reminding the developer to add a new procedure while removing an existing procedure by changing it into a comment:

```
public void myMethod()
{
    //Declare variables
    int value;

//TODO Validate if calculation is really required
/*
    //Perform calculation
    value = this.calc();
*/
    ...
}
```

XML documentation

You can document XML methods and classes directly in X++ by typing three slash characters (///) followed by structured documentation in XML format. The XML documentation must be above the actual code.

The XML documentation must align with the code. The Best Practices tool contains a set of rules that can validate the XML documentation. Table 4-12 lists the supported tags.

TABLE 4-12 XML tags supported for XML documentation.

Tag	Description
<*summary*>	Describes a method or a class
<*param*>	Describes the parameters of a method
<*returns*>	Describes the return value of a method
<*remarks*>	Adds information that supplements the information provided in the <*summary*> tag
<*exception*>	Documents exceptions that are thrown by a method
<*permission*>	Describes the permission needed to access methods using *CodeAccessSecurity.demand*
<*seealso*>	Lists references to related and relevant *documentation*

The XML documentation is automatically displayed in the IntelliSense in the X++ editor.

You can extract the written XML documentation for an AOT project by using the Add-Ins menu option Extract XML Documentation. One XML file is produced that contains all of the documentation for the elements inside the project. You can also use this XML file to publish the documentation.

The following code example shows XML documentation for a static method on the *Global* class:

```
/// <summary>
/// Converts an X++ utcDateTime value to a .NET System.DateTime object.
/// </summary>
/// <param name="_utcDateTime">
/// The X++ utcDateTime to convert.
/// </param>
/// <returns>
/// A .NET System.DateTime object.
/// </returns>
static client server anytype utcDateTime2SystemDateTime(utcDateTime _utcDateTime)
{
    return CLRInterop::getObjectForAnyType(_utcDateTime);
}
```

Classes and interfaces

You define types and their structure in the AOT, not in the X++ language. Other programming languages that support type declarations do so within code, but Microsoft Dynamics AX supports an object layering feature that accepts X++ source code customizations to type declaration parts that encompass variable declarations and method declarations. Each part of a type declaration is managed as a separate compilation unit, and model data is used to manage, persist, and reconstitute dynamic types whose parts can include compilation units from many object layers.

You use X++ to define logic, including method profiles (return value, method name, and parameter type and name). You use the X++ editor to add new methods to the AOT, so you can construct types without leaving the X++ editor.

You use X++ class declarations to declare protected instance variable fields that are members of application logic and framework reference types. You can't declare private or public variable fields. You can declare classes as abstract if they are incomplete type specifications that can't be instantiated. You can also declare them final if they are complete specifications that can't be further specialized. The following code provides an example of an abstract class declaration header:

```
abstract class MyClass
{
}
```

You can also structure classes into single-inheritance generalization or specialization hierarchies in which derived classes inherit and override members of base classes. The following code shows an example of a derived class declaration header that specifies that *MyDerivedClass* extends the abstract base class *MyClass*. It also specifies that *MyDerivedClass* is final and can't be further specialized by another class. Because X++ doesn't support multiple inheritance, derived classes can extend only one base class.

```
final class MyDerivedClass extends MyClass
{
}
```

X++ also supports interface type specifications that specify method signatures but don't define their implementation. Classes can implement more than one interface, but the class and its derived classes should together provide definitions for the methods declared in all the interfaces. If it fails to provide the method definitions, the class itself is treated as abstract and cannot be instantiated. The following code provides an example of an interface declaration header and a class declaration header that implements the interface:

```
interface MyInterface
{
    void myMethod()
    {
    }
}
class MyClass implements MyInterface
{
    void myMethod()
    {
    }
}
```

Fields

A field is a class member that represents a variable and its type. Fields are declared in class declaration headers; each class and interface has a definition part with the name *classDeclaration* in the AOT. Fields are accessible only to code statements that are part of the class declaration or derived class declarations. Assignment statements are not allowed in class declaration headers. The following example demonstrates how variables are initialized with assignment statements in a *new* method:

```
class MyClass
{
    str s;
    int i;
    MyClass1 myClass1;

    public void new()
    {
        i = 0;
        myClass1 = new MyClass1();
    }
}
```

Methods

A method on a class is a member that uses statements to define the behavior of an object. An interface method is a member that declares an expected behavior of an object. The following code provides an example of a method declaration on an interface and an implementation of the method on a class that implements the interface:

```
interface MyInterface
{
    public str myMethod()
    {
    }
}
```

```
class MyClass implements MyInterface
{
    public str myMethod();
    {
        return "Hello World";
    }
}
```

Methods are defined with public, private, or protected access modifiers. If an access modifier is omitted, the method is publicly accessible. The X++ template for new methods provides the private access specifier. Table 4-13 describes additional method modifiers supported by X++.

TABLE 4-13 Method modifiers supported by X++.

Modifier	Description
abstract	Abstract methods have no implementation. Derived classes must provide definitions for abstract methods.
client	Client methods can execute only on a MorphX client. The *client* modifier is allowed only on static methods.
delegate	Delegate methods cannot contain implementation. Event handlers can subscribe to delegate methods. The *delegate* modifier is allowed only on instance methods.
display	Display methods are invoked each time a form is redrawn. The *display* modifier is allowed only on table, form, form data source, and form control methods.
edit	The *edit* method is invoked each time a form is redrawn or a user provides input through a form control. The *edit* modifier is allowed only on table, form, and form data source methods.
final	Final methods can't be overridden by methods with the same name in derived classes.
server	Server methods can execute only on an Application Object Server (AOS). The *server* modifier is allowed on all table methods and on static class methods.
static	Static methods are called using the name of the class rather than the name of an instance of the class. Fields can't be accessed from within a static method.

Method parameters can have default values that are used when parameters are omitted from method invocations. The following code sample prints "Hello World" when *myMethod* is invoked with no parameters:

```
public void myMethod(str s = "Hello World")
{
    print s;
    pause;
}

public void run()
{
    this.myMethod();
}
```

A constructor is a special instance method that is invoked to initialize an object when the *new* operator is executed by the Microsoft Dynamics AX runtime. You can't call constructors directly from X++ code. The sample on the next page provides an example of a class declaration header and an instance constructor method that takes one parameter as an argument.

```
class MyClass
{
    int i;

    public void new(int _i)
    {
        i = _i;
    }
}
```

Delegates

The purpose of delegates is to expose extension points where add-ons and customizations can extend the application in a lightweight manner without injecting logic into the base functionality. Delegates are methods without any implementation. Delegates are always public and cannot have a return value. You declare a delegate using the *delegate* keyword. You invoke a delegate using the same syntax as a standard method invocation:

```
class MyClass
{
    delegate void myDelegate(int _i)
    {
    }

    private void myMethod()
    {
        this.myDelegate(42);
    }
}
```

When a delegate is invoked, the runtime automatically invokes all event handlers that subscribe to the delegate. There are two ways of subscribing to delegates: declaratively and dynamically. The runtime does not define the sequence in which event handlers are invoked. If your logic relies on an invocation sequence, you should use mechanisms other than delegates and event handlers.

To subscribe declaratively, right-click a delegate in the AOT and then select New Event Handler Subscription. On the resulting event handler node in the AOT, you can specify the class and the static method that will be invoked. The class can be either an X++ class or a .NET class.

To subscribe dynamically, you use the keyword *eventhandler*. Notice that when subscribing dynamically, the event handler is an instance method. It is also possible to unsubscribe.

```
class MyEventHandlerClass
{
    public void myEventHandler(int _i)
    {
        ...
    }

    public static void myStaticEventHandler(int _i)
    {
        ...
    }

    public static void main(Args args)
    {
        MyClass myClass = new MyClass();
        MyEventHandlerClass myEventHandlerClass = new MyEventHandlerClass();

        //Subscribe
        myClass.myDelegate += eventhandler(myEventHandlerClass.myEventHandler);
        myClass.myDelegate +=
                        eventhandler(MyEventHandlerClass::myStaticEventHandler);

        //Unsubscribe
        myClass.myDelegate -= eventhandler(myEventHandlerClass.myEventHandler);
        myClass.myDelegate -=
                        eventhandler(MyEventHandlerClass::myStaticEventHandler);
    }
}
```

Regardless of how you subscribe, the event handler must be public, return *void*, and have the same parameters as the delegate.

Note Cross-tier events are not supported.

As an alternative to delegates, you can achieve a similar effect by using pre- and post-event handlers.

Pre- and post-event handlers

You can subscribe declaratively to any class and record type method by using the same procedure as for delegates. The event handler is invoked either before or after the method is invoked. Event handlers for pre- and post-methods must be public, static, void, and either take the same parameters as the method or one parameter of the *XppPrePostArgs* type.

The simplest, type-safe implementation uses syntax where the parameters of the method and the event handler method match.

```
class MyClass
{
    public int myMethod(int _i)
    {
        return _i;
    }
}

class MyEventHandlerClass
{
    public static void myPreEventHandler(int _i)
    {
        if (_i > 100)
        {
            ...
        }
    }

    public static void myPostEventHandler(int _i)
    {
        if (_i > 100)
        {
            ...
        }
    }
}
```

If you need to manipulate either the parameters or the return value, the event handler must take one parameter of the *XppPrePostArgs* type.

To create such an event handler, right-click the class, and then select New pre- or post-event handler. The *XppPrePostArgs* class provides access to the parameters and the return values of the method. You can even alter parameter values in pre-event handlers and alter the return value in post-event handlers.

```
class MyClass
{
    public int myMethod(int _i)
    {
        return _i;
    }
}

class MyEventHandlerClass
{
    public static void myPreEventHandler(XppPrePostArgs _args)
    {
        if (_args.existsArg('_i') &&
            _args.getArg('_i') > 100)
        {
            _args.setArg('_i', 100);
        }

    }

    public static void myPostEventHandler(XppPrePostArgs _args)
    {
        if (_args.getReturnValue() < 0)
        {
            _args.setReturnValue(0);
        }
    }
}
```

Attributes

Classes and methods can be decorated with attributes to convey declarative information to other code, such as the runtime, the compiler, frameworks, or other tools. To decorate the class, you insert the attribute in the *classDeclaration* element. To decorate a method, you insert the attribute before the method declaration:

```
[MyAttribute("Some parameter")]
class MyClass
{
    [MyAttribute("Some other parameter")]
    public void myMethod()
    {
        ...
    }
}
```

The first attribute that was built in Microsoft Dynamics AX 2012 was the *SysObsoleteAttribute* attribute. By decorating a class or a method with this attribute, any consuming code is notified during compilation that the target is obsolete. You can create your own attributes by creating classes that extend the *SysAttribute* class:

```
class MyAttribute extends SysAttribute
{
    str parameter;

    public void new(str _parameter)
    {
        parameter = _parameter;
        super();
    }
}
```

Code access security

Code access security (CAS) is a mechanism designed to protect systems from dangerous APIs that are invoked by untrusted code. CAS has nothing to do with user authentication or authorization; it is a mechanism allowing two pieces of code to communicate in a manner that cannot be compromised.

Caution X++ developers are responsible for writing code that conforms to Trustworthy Computing guidelines. You can find those guidelines in the white paper "Writing Secure X++ Code," available from the Microsoft Dynamics AX Developer Center (*http://msdn.microsoft.com/en-us/dynamics/ax*).

In the Microsoft Dynamics AX implementation of CAS, trusted code is defined as code from the AOT running on the Application Object Server (AOS). The first part of the definition ensures that the code is written by a trusted X++ developer. Developer privileges are the highest level of privileges in Microsoft Dynamics AX and should be granted only to trusted personnel. The second part of the definition ensures the code that the trusted developer has written hasn't been tampered with. If the code executes outside the AOS—on a client, for example—it can't be trusted because of the possibility that it was altered on the client side before execution. Untrusted code also includes code that is executed through the *runBuf* and *evalBuf* methods. These methods are typically used to execute code generated at run time based on user input.

CAS enables a secure handshake between an API and its consumer. Only consumers who provide the correct handshake can invoke the API. Any other invocation raises an exception.

The secure handshake is established through the *CodeAccessPermission* class or one of its specializations. The consumer must request permission to call the API, which is done by calling *CodeAccessPermission.assert*. The API verifies that the consumer has the correct permissions by calling *CodeAccessPermission.demand*. The *demand* method searches the call stack for a matching assertion. If untrusted code exists on the call stack before the matching assertion, an exception is raised. This process is illustrated in Figure 4-1.

The following code contains an example of a dangerous API protected by CAS and a consumer providing the correct permissions to invoke the API:

```
class WinApiServer
{
    // Delete any given file on the server
    public server static boolean deleteFile(Filename _fileName)
    {
        FileIOPermission    fileIOPerm;

        // Check file I/O permission
        fileIOPerm = new FileIOPermission(_fileName, 'w');
        fileIOPerm.demand();

        // Delete the file

        System.IO.File::Delete(_filename);
    }
}

class Consumer
{
    // Delete the temporary file on the server
    public server static void deleteTmpFile()
    {
        FileIOPermission    fileIOPerm;
        FileName            filename = @'c:\tmp\file.tmp';

        // Request file I/O permission
        fileIOPerm = new FileIOPermission(filename, 'w');
        fileIOPerm.assert();

        // Use CAS protected API to delete the file
        WinApiServer::deleteFile(filename);
    }
}
```

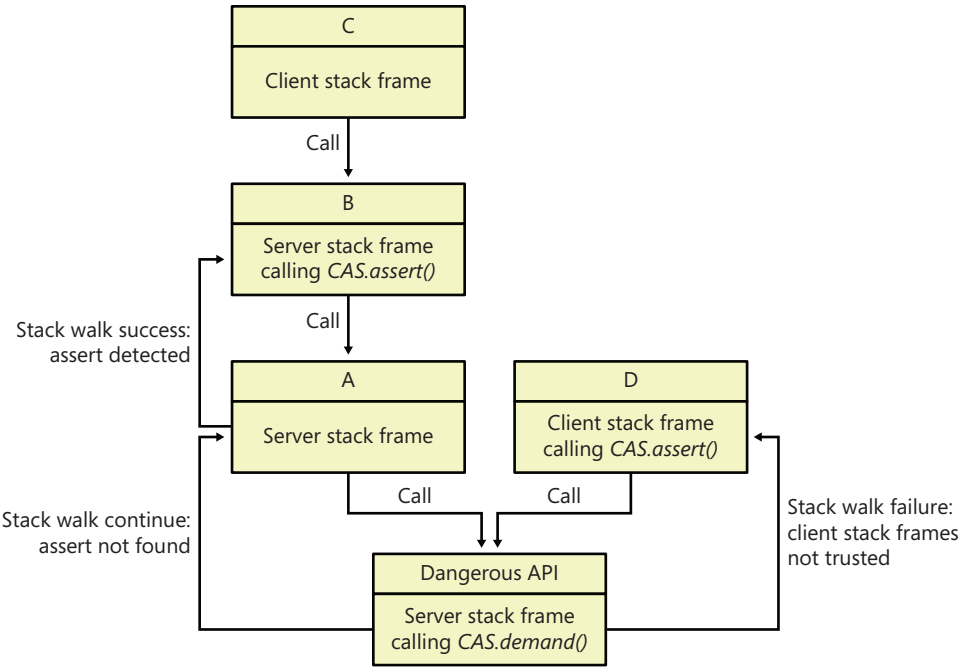


FIGURE 4-1 CAS stack frame walk.

WinAPIServer::deleteFile is considered to be a dangerous API because it exposes the .NET API *System.IO.File::Delete(string fileName)*. Exposing this API on the server is dangerous because it allows the user to remotely delete files on the server, possibly bringing the server down. In the example, *WinApiServer::deleteFile* demands that the caller has asserted that the input file name is valid. The demand prevents use of the API from the client tier and from any code not stored in the AOT.

Caution When using *assert*, make sure that you don't create a new API that is just as dangerous as the one that CAS has secured. When you call *assert*, you are asserting that your code doesn't expose the same vulnerability that required the protection of CAS. For example, if the *deleteTmpFile* method in the previous example had taken the file name as a parameter, it could have been used to bypass the CAS protection of *WinApi::deleteFile* and delete any file on the server.

Compiling and running X++ as .NET CIL

All X++ code is compiled into Microsoft Dynamics AX runtime bytecode intermediate format. This format is used by the Microsoft Dynamics AX runtime for Microsoft Dynamics AX client and server code.

Further, classes and tables are compiled into .NET common intermediate language (CIL). This format is used by X++ code executed by the Batch Server and in certain other scenarios.

The X++ compiler only generates Microsoft Dynamics AX runtime bytecode to generate CIL code; you must manually press either the Generate Full IL or Generate Incremental IL button. Both are available on the toolbar.

The main benefit of running X++ as CIL is performance. Generally the .NET runtime is significantly faster than the X++ runtime. In certain constructions, the performance gain is particularly remarkable:

- **Constructs with many method calls—**Under the covers in the X++ runtime, any method call happens through reflection, whereas in CIL, this happens at the CPU level.
- **Constructions with many short-lived objects—**Garbage collection in the Microsoft Dynamics AX runtime is deterministic, which means that whenever an instance goes out of scope, the entire object graph is analyzed to determine if any objects can be deallocated. In the .NET CLR, garbage collection is indeterministic, which means that the runtime determines the optimal time for reclaiming memory.
- **Constructions with extensive use of.NET interop—**When running as X++ code as CIL, all conversion and marshaling between the runtimes are avoided.

Note The capability to compile X++ into CIL requires that X++ syntax be as strict as the syntax in managed code. The most noteworthy change is that overridden methods must now have the same signature as the base method. The only permissible discrepancy is the addition of optional parameters.

One real-life example where running X++ code as .NET CIL makes a significant difference is in the compare tool. The compare algorithm is implemented as X++ code in the *SysCompareText* class. Even though the algorithm has few method calls, few short-lived objects, and no .NET interop, the switch to CIL means that within a time frame of 10 seconds, it is now possible to compare two 3,500-line texts, whereas the AX runtime can only handle 600 lines in the same time frame. The complexity of the algorithm is exponential. In other words the performance gain gets even more significant the larger the texts become.

All services and batch jobs will automatically run as CIL. If you want to force X++ code to run as CIL in non-batch scenarios, you use the methods *runClassMethodIL* and *runTableMethodIL* on the *Global* class. The IL entry point must be a static server method that returns a container and takes one container parameter:

```
class MyClass
{
    private static server container addInIL(container _parameters)
    {
        int p1, p2;
        [p1, p2] = _parameters;
```

```
        return [p1+p2];
    }

    public server static void main(Args _args)
    {
        int result;
        XppILExecutePermission permission = new XppILExecutePermission();
        permission.assert();
        [result] = runClassMethodIL(classStr(MyClass),
                                    staticMethodStr(MyClass, addInIL), [2, 2]);
        info(strFmt("The result from IL is: %1", result));
    }
}
```

Design and implementation patterns

So far, this chapter has described the individual elements of X++. You've seen that statements are grouped into methods, and methods are grouped into classes, tables, and other model element types. These structures enable you to create X++ code at a higher level of abstraction. The following example shows how an assignment operation can be encapsulated into a method to clearly articulate the intention of the code.

```
control.show();
```

is at a higher level of abstraction than

```
flags = flags | 0x0004;
```

By using patterns, developers can communicate their solutions more effectively and reuse proven solutions to common problems. Patterns help readers of source code to quickly understand the purpose of a particular implementation. Bear in mind that even as a code author, you spend more time reading source code than writing it.

Implementations of patterns are typically recognizable by the names used for classes, methods, parameters, and variables. Arguably, naming these elements so that they effectively convey the intention of the code is the developer's most difficult task. Much of the information in existing literature on design patterns pertains to object-oriented languages, and you can benefit from exploring that information to find patterns and techniques you can apply when you're writing X++

code. Design patterns express relationships or interactions between several classes or objects. They don't prescribe a specific implementation, but they do offer a template solution for a typical design problem. In contrast, implementation patterns are implementation specific and can have a scope that spans only a single statement.

This section highlights some of the most frequently used patterns specific to X++. More descriptions are available in the Microsoft Dynamics AX SDK on MSDN.

Class-level patterns

These patterns apply to classes in X++.

Parameter method

To set and get a class field from outside the class, you should implement a parameter method. The parameter method should have the same name as the field and be prefixed with *parm*. Parameter methods come in two flavors: *get-only* and *get/set*.

```
public class Employee
{
    EmployeeName name;

    public EmployeeName parmName(EmployeeName _name = name)
    {
        name = _name;
        return name;
    }
}
```

Constructor encapsulation

The purpose of the constructor encapsulation pattern is to enable Liskov's class substitution principle. In other words, with constructor encapsulation, you can replace an existing class with a customized class without using the layering system. Just as in the layering system, this pattern enables changing the logic in a class without having to update any references to the class. Be careful to avoid overlayering because it often causes upgrade conflicts.

Classes that have a static *construct* method follow the constructor encapsulation pattern. The *construct* method should instantiate the class and immediately return the instance. The *construct* method must be static and shouldn't take any parameters.

When parameters are required, you should implement the static *new* methods. These methods call the *construct* method to instantiate the class and then call the parameter methods to set the parameters. In this case, the *construct* method should be private:

```
public class Employee
{
    ...
    protected void new()
    {
    }

    protected static Employee construct()
    {
        return new Employee();
    }

    public static Employee newName(EmployeeName name)
    {
        Employee employee = Employee::construct();

        employee.parmName(name);
        return employee;
    }
}
```

Factory

To decouple a base class from derived classes, use the *SysExtension* framework. This framework enables the construction of an instance of a class based on its attributes. This pattern enables add-ons and customizations to add new subclasses without touching the base class or the factory method:

```
class BaseClass
{
    ...
    public static BaseClass newFromTableName(TableName _tableName)
    {
        SysTableAttribute attribute = new SysTableAttribute(_tableName);

        return SysExtensionAppClassFactory::getClassFromSysAttribute(
            classStr(BaseClass), attribute);
    }
}

[SysTableAttribute(tableStr(MyTable))]
class Subclass extends BaseClass
{
   ...
}
```

Serialization with the *pack* and *unpack* methods

Many classes require the capability to serialize and deserialize themselves. Serialization is an operation that extracts an object's state into value-type data; deserialization creates an instance from that data.

X++ classes that implement the *Packable* interface support serialization. The *Packable* interface contains two methods: *pack* and *unpack*. The *pack* method returns a container with the object's state; the *unpack* method takes a container as a parameter and sets the object's state accordingly. You should include a versioning number as the first entry in the container to make the code resilient to old packed data stored in the database when the implementation changes.

```
public class Employee implements SysPackable
{
    EmployeeName name;
    #define.currentVersion(1)
    #localmacro.CurrentList
        name
    #endmacro
    ...

    public container pack()
    {
        return [#currentVersion, #currentList];
    }

    public boolean unpack(container packedClass)
    {
        Version version = RunBase::getVersion(packedClass);

        switch (version)
        {
            case #CurrentVersion:
                [version, #CurrentList] = packedClass;
                break;
            default:   //The version number is unsupported
                return false;
        }
        return true;
    }
}
```

Table-level patterns

The patterns described in this section—the *find* and *exists* methods, polymorphic associations (Table/Group/All), and Generic Record References—apply to tables.

Find and e*xists* methods

Each table must have the two static methods *find* and *exists*. They both take the primary keys of the table as parameters and return the matching record or a Boolean value, respectively. Besides the primary keys, the *Find* method also takes a *Boolean* parameter that specifies whether the record should be selected for update.

For the *CustTable* table, these methods have the following profiles:

```
static CustTable find(CustAccount _custAccount, boolean _forUpdate = false)
static boolean exist(CustAccount _custAccount)
```

Polymorphic associations

The Table/Group/All pattern is used to model a polymorphic association to either a specific record in another table, a collection of records in another table, or all records in another table. For example, a record could be associated with a specific item, all items in an item group, or all items.

You implement the Table/Group/All pattern by creating two fields and two relations on the table. By convention, the name of the first field has the suffix *Code*; for example, *ItemCode*. This field is modeled using the base enum *TableGroupAll*. The name of the second field usually has the suffix *Relation*; for example, *ItemRelation*. This field is modeled by using the extended data type that is the primary key in the foreign tables. The two relations are of the type *Fixed* field relation. The first relation specifies that when the *Code* field equals 0 (*TableGroupAll::Table*), the *Relation* field equals the primary key in the foreign master data table. The second relation specifies that when the *Code* field equals 1 (*TableGroupAll::Group*), the *Relation* field equals the primary key in the foreign grouping table.

Figure 4-2 shows an example.

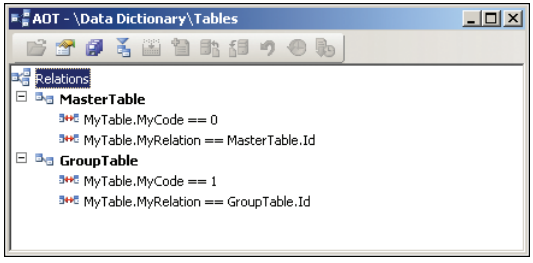


FIGURE 4-2 A polymorphic association.

Generic record reference

The Generic Record Reference pattern is a variation of the Table/Group/All pattern. This pattern is used to model an association to a foreign table. It comes in three flavors: (a) an association to any record in a specific table, (b) an association to any record in a fixed set of specific tables, and (c) an association to any record in any table.

All three flavors of this pattern are implemented by creating a field that uses the *RefRecId* extended data type.

To model an association to any record in a specific table (flavor a), a relation is created from the *RefRecId* field to the *RecId* field of the foreign table, as illustrated in Figure 4-3.

FIGURE 4-3 An association to a specific table.

For flavors b and c, an additional field is required. This field is created by using the *RefTableId* extended data type. To model an association to any record in a fixed set of specific tables (flavor b), a relation is created for each foreign table from the *RefTableId* field to the *TableId* field of the foreign table, and from the *RefRecId* field to the *RecId* field of the foreign table, as shown in Figure 4-4.

FIGURE 4-4 An association to any record in a fixed set of tables.

To model an association to any record in any table (flavor c), a relation is created from the *RefTableId* field to the generic table *Common TableId* field and from the *RefRecId* field to *Common RecId* field, as shown in Figure 4-5.

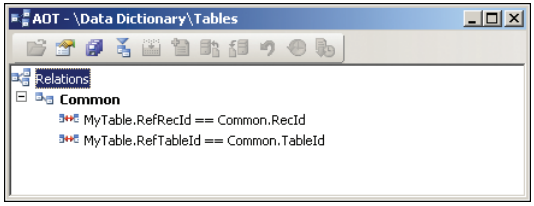


FIGURE 4-5 An association to any record in any table.

PART II

Developing with Microsoft Dynamics AX

CHAPTER 5 Designing the user experience **137**

CHAPTER 6 The Microsoft Dynamics AX client **159**

CHAPTER 7 Enterprise Portal. **195**

CHAPTER 8 Workflow in Microsoft Dynamics AX **245**

CHAPTER 9 Reporting **275**

CHAPTER 10 BI and analytics **299**

CHAPTER 11 Security, licensing, and configuration **351**

CHAPTER 12 Microsoft Dynamics AX services **and** integration. **385**

CHAPTER 13 Performance **417**

CHAPTER 14 Extending Microsoft Dynamics AX. **493**

CHAPTER 15 Testing **527**

CHAPTER 16 Customizing and adding help **545**

CHAPTER 5

Designing the user experience

In this chapter

Introduction 277
A role-tailored design approach 279
User experience components 280
Role Center pages 282
Area pages 284
List pages 286
Details forms 290
Transaction details forms 293
Enterprise Portal web client user experience 295
Design for your users 297

Introduction

Microsoft Dynamics AX 2012 has been marketed as "Powerfully Simple." Making this a reality was not just a marketing slogan, but instead was a key design goal for the release.

As an enterprise resource planning (ERP) solution, Microsoft Dynamics AX must provide the many powerful, built-in capabilities that are required to run a thriving company in the twenty-first century. The needs of organizations are becoming more complex. Companies are trying to organize themselves in new and unique ways to become more efficient. Leaders of these organizations are asking their people to achieve more with less. Governments want more transparency in the business operations of a company. Combined, all these things increase the complexity of running a business and the demands on an ERP system.

The challenge for Microsoft Dynamics AX 2012 was to harness these powerful capabilities in a way that users would find simple to use. There is a natural tension between these goals, but that tension is far from irreconcilable.

At Microsoft, simplicity is defined as the reduction or elimination of an attribute of the design that target users are aware of or consider unessential. The easiest way to simplify a design is by removing elements from that design. For example, if you want to simplify the experience of creating a new customer, you can do so easily by reducing the number fields that the user needs to complete on the new customer form. With fewer fields, the user can complete the form in fewer keystrokes, which also minimizes the chance that the user will make a mistake. The problem is that fields can't simply be removed from the new customer form because those fields are required to support the powerful capabilities that customers need.

So to have a simple and powerful user experience, Microsoft Dynamics AX 2012 has been designed for the *probable*, not the possible. To design for the probable means that you need to truly understand what the user is most likely to do and not assume that all actions are equally possible. You can focus your designs on what is likely, and then reduce, hide, or remove what is unlikely.

For example, Microsoft Dynamics AX contains approximately 100 fields that contain information about a customer. In the prior release, when the user created a new customer, the Customer Details form presented all 100 fields. The user had to look through all these possible fields to determine what to enter. Microsoft Dynamics AX 2012 introduced a new dialog box (Figure 5-1) that appears when a user creates a new customer. This dialog box displays the 25 fields that users are most likely to use. The user can simply enter data in these fields, and then click Save And Close to create the new customer. If the user needs to enter more detailed information about the customer, he or she can click Save And Open to go to the full Customer Details form to enter data in the other 75 fields.

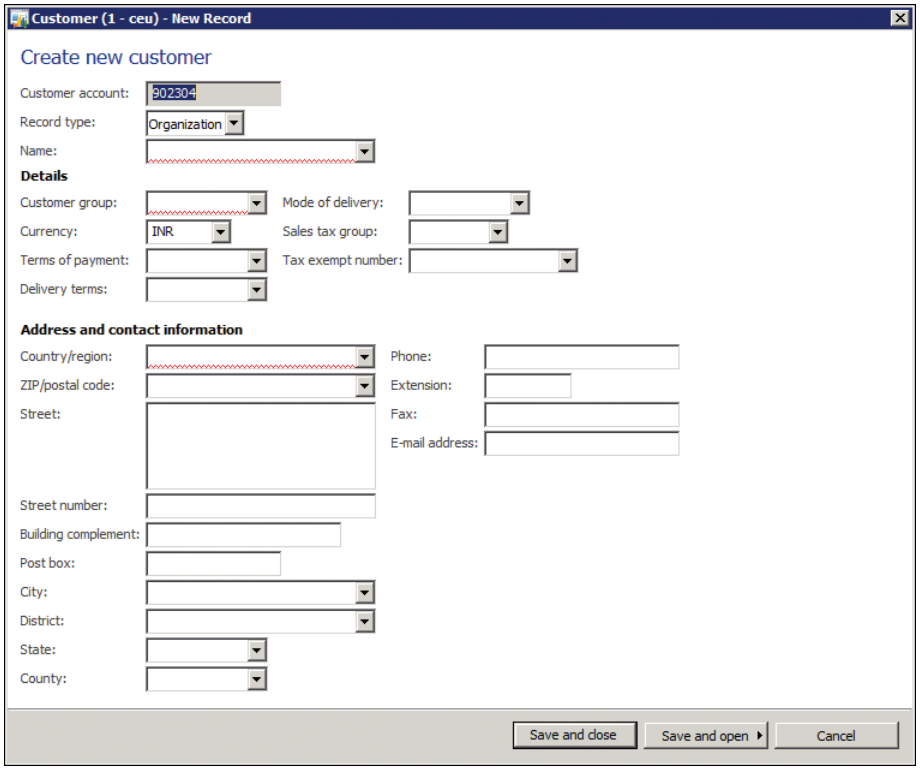


FIGURE 5-1 Simplified Customer dialog box.

This chapter describes the key concepts of the Microsoft Dynamics AX user experience and explains how you can extend the capabilities of the product while maintaining a focus on simplicity.

This chapter supplements the Microsoft Dynamics AX 2012 User Experience Guidelines on MSDN (*http://msdn.microsoft.com/en-us/library/gg886610.aspx*). For more detailed information, refer to these guidelines.

A role-tailored design approach

Designing an ERP system that is simple for all users is challenging because many types of users use the product. The pool of users encompasses more than 86 roles, and those roles use Microsoft Dynamics AX for many different scenarios. These scenarios range from picking and packing items in a warehouse to processing payments from a customer in the finance department. It is not surprising that users in these various roles have different mental models for how the system should work for them. Designing the user experience for specific roles provides a much better experience than providing the same experience for all users.

Historically, ERP systems were designed as a thin wrapper around the tables in the database. If the database table had 20 fields, the user interface would display those 20 fields on a single form, similar to how they were stored in the database. When a new feature was needed, new fields were added to the table and those fields were displayed on the form. Over time, ERP systems became very complex because more and more fields were added without regard to who would be using them. This led to user experiences that were designed for everyone but optimized for no one. The end goal of a role-tailored user experience in Microsoft Dynamics AX is to make the user feel as if the system was designed for him or her.

In Microsoft Dynamics AX 2012, the user experience is tailored for the various roles that the product targets. The security system includes 86 roles that system administrators can assign to specific groups of users. The user experience is tailored automatically based on the roles and shows only the content that is needed by a user who belongs to a given role. Based on the user's role, actions on the Action pane, fields, field groups, or entire tabs might be removed from certain forms. With each field or button that is hidden, the product becomes easier to use. The menu structure is also tailored so that each user sees only the areas or the content in these areas that pertain to the user's role. Users feel like they are using a smaller application tailored to their needs, as opposed to a large, monolithic ERP system. For more information about working with roles, see Chapter 11, "Security, licensing, and configuration."

To illustrate this concept, look at the navigational structure for Microsoft Dynamics AX 2012. The product contains 20 area pages targeting the various activities needed to run a business. While a system administrator sees all these areas, a specific role such as a Shipping Clerk, Purchasing Agent, or Order Processor sees only the four to six areas that relate to the role, as shown in Figure 5-2.

FIGURE 5-2 Role-tailored navigation.

User experience components

In Microsoft Dynamics AX, user experience components are divided into two conceptual layers:

- The *navigation layer* consists of top-level pages that serve as a starting point for user as he or she navigates through the application. Area pages, Role Centers, and list pages are navigation-layer elements.
- The *work layer* consists of the forms in which users perform their daily work, such as creating and editing record and entering and processing transactions. Details forms and transaction details forms are work-layer elements.

Figure 5-3 illustrates how the user navigates through the primary elements that make up the Microsoft Dynamics AX 2012 user experience. The following sections describe these elements in detail.

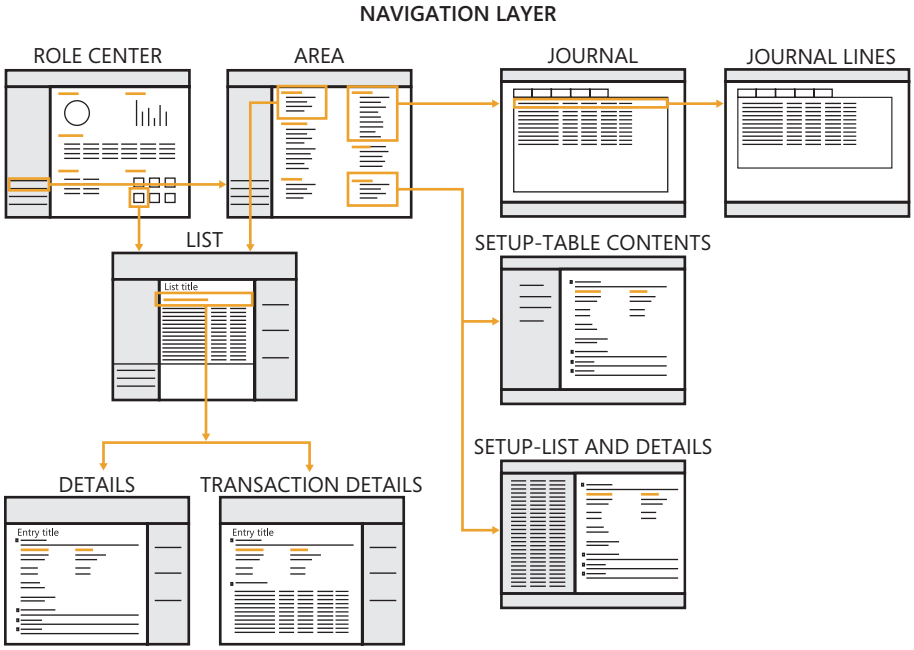


FIGURE 5-3 Navigation paths through Microsoft Dynamics AX 2012.

Navigation layer forms

Navigation layer forms such as the Role Centers, area pages, and list pages are displayed within the Microsoft Dynamics AX Windows client in a flat navigation model. This model is similar to that of a website, in that pages are displayed within the content region of the page, replacing each other as the user progresses from one form to the next. The client workspace consists of the following components, as illustrated in Figure 5-4:

- **Address bar** Provides an alternate method of navigating through the application. A user can type a path or click the arrow icon next to each entry in the path to select the next location. The address bar has buttons that allow navigation backward and forward between the recently displayed pages. The address bar also provides a mechanism for the user to switch companies because the current company is the first entry in the address path.
- **Search bar** Lets users search for data, menu items, or help content. The user can use the search bar as an alternate method of navigation if he or she doesn't know how to find a particular form. The search bar is an optional component that must be configured as part of setup. For more information, see "Enterprise Search" at *http://technet.microsoft.com/en-us/library/gg731850.aspx*.
- **Navigation pane** Appears on the left edge of the client workspace and is used for navigating to the various areas within the application or the user's list of favorite forms. Optionally, this pane can be collapsed or hidden through the View menu.

- **Content pane** Appears to the right of the navigation pane and displays top-level pages, such as area pages, Role Centers, and list pages.
- **FactBox pane** Appears at the right of the workspace and provides related information about a specific record in a grid. The FactBox pane appears only on list pages. Users can personalize the contents of the FactBox pane by using the View menu.
- **Status bar** Appears at the bottom of the workspace and displays additional information in a consistent location, such as user name, company, or notifications. The user can personalize the contents of the status bar by using the Options form (Click File > Tools > Options).

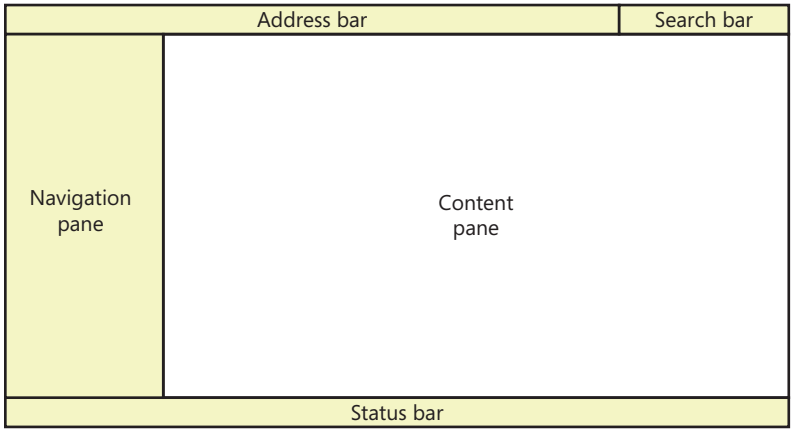


FIGURE 5-4 Client workspace components.

Work layer forms

The remaining forms in Microsoft Dynamics AX are where the user performs work such as configuring the system, creating new transactions, or entering information into journals. These forms open in a new window that is separate from the client workspace. The work layer pages are described in detail in the upcoming sections.

Role Center pages

A Role Center page is the user's home page in the application. A Role Center provides a dashboard of information that pertains to a user's job function in the business or organization. This information includes transaction data, alerts, links, and common tasks that are associated with the user's role in the company.

Microsoft Dynamics AX 2012 provides different Role Center content for the various roles. Each Role Center provides the information that the users who belong to that role need to monitor their work. A Role Center also provides shortcuts to frequently used data and forms. Each user can personalize the content that appears in his or her Role Center.

Cues

A cue is a visual representation of a query that appears as a stack of paper. A cue represents the activities that the user needs to perform. The stack grows and shrinks as the results of the query change.

Cues are an excellent way for users to monitor their work. For example, an Accounts Payable clerk can monitor a cue of pending invoices, invoices due today, or invoices past due, as shown in Figure 5-5. Clicking a cue opens the appropriate list page with the same query applied. When the clerk wants to act on the invoices, the clerk clicks the cue.

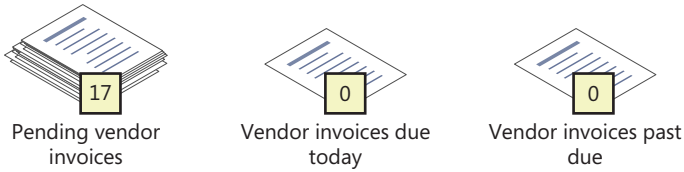


FIGURE 5-5 Activity cues.

Design Role Centers

While Microsoft Dynamics AX includes great Role Centers for the various roles, they must be customized to meet the needs of the people who will be using them. It is highly recommended that partners and system administrators take the time to customize the Role Centers for the various users within the organization.

Designing a great Role Center requires a deep understanding of the user. Here are a few techniques that you can use to help understand your customers:

- Survey people in the various roles to understand the top 10 questions they have related to their job. Then, explore ways that you can provide answers to as many of those questions as possible with a Role Center.
- Show users the content of their default Role Center on a piece of paper. Then, ask them to circle the content that they find useful and to cross out the content that they don't find useful. You can also give them a blank piece of paper to sketch out additional content that they would like to see. Users typically get excited with these types of exercises because they feel empowered describing what they want from their ERP system.
- Observe users performing their daily tasks. Often, users cannot articulate what they need to become more efficient, but it may be obvious if you observe them performing their jobs. As you are observing them, watch for the patterns that emerge in their work.
- Find out which forms users open frequently, and consider adding links to those forms to the Role Center QuickLink on their Role Center or a favorite in their navigation pane. A QuickLink is a part on the Role Center that provides quick access to any form within Microsoft Dynamics AX.

- Find out if users frequently go to a list page and filter the content to see a particular group of records. If so, you can make them more efficient by adding a cue to the Role Center that is configured to provide direct access to this list with the appropriate filter applied. For some users, we've seen Role Centers that have been customized to include a page full of cues that were needed by the user.
- Summarize frequently viewed reports as a chart or graph.

Here are a few other things to consider when you design a new Role Center or extend an existing Role Center:

- Remove any parts that users don't need.
- Place the most important content toward the top of the page.
- Ensure that the page loads quickly, within 2 to 5 seconds. This may require you to optimize the queries and cubes that display the information within these parts. For more information about optimizing queries, see Chapter 13, "Performance."

Area pages

Area pages are the primary method for users to navigate through the application. By default, Microsoft Dynamics AX provides 20 different area pages. Each area page focuses on a specific department or activity; for example, Human Resources or Accounts Receivable, as shown in Figure 5-6. Depending on their role, users may see only a small set of area pages.

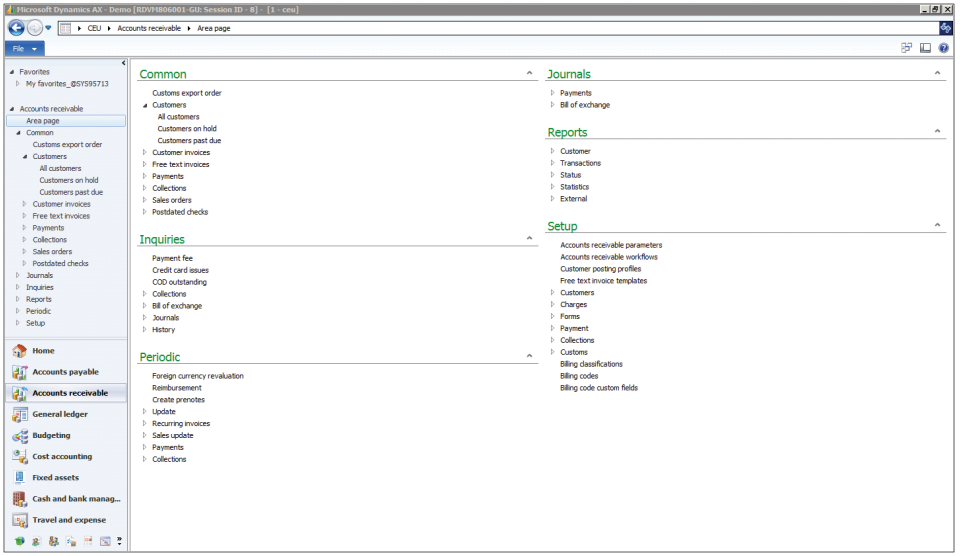


FIGURE 5-6 Area page for Accounts Receivable.

The content of an area page is divided into six groups of links:

- **Common** Contains links to the most important entities that are used within this area, such as customers, vendors, products, sales orders, invoices, and so on. These links usually take the user to the list page for an entity. Through the list page, users should be able to navigate to all things that are related to the entity.
- **Inquiries** Provides access to all the inquiry-type forms for the area. If possible, do not create new inquiry forms for entities that have list pages. Instead, consider providing different views within a list page because lists are where users expect for find all content related to an entity.
- **Periodic** Provides access to tasks that need to be performed periodically. If you are considering adding new forms to the Periodic section, think about whether the form is specific to an entity that would be better suited to be accessed through the entity's list page and details page.
- **Journals** Provides access to journals that are related to this area. A journal is a concept that makes sense to a financial user, but not to other users outside the finance department. Use caution when you introduce new journals to Microsoft Dynamics AX to make sure that this is the correct approach for your users.
- **Reports** Provides access to reports that are related to this area. Note that we are discouraging creating new reports whenever possible. Users don't want to view information in a report, but instead prefer seeing this information on a form such as a list page because it is more interactive than a report.
- **Setup** Provides links to the forms needed to configure this area.

Design area pages

Designing area pages is an exercise in organizing the content in a way that makes sense for users. Here are some tips to consider when you design a new area page or extend an existing area page.

Take the time to understand the user's mental model as it relates to the work that they do. Make sure that you place the links to the forms they need to use in the most logical area. The best way to do this is to perform a simple card sort exercise to help you understand how users want to organize their content.

To conduct a card sort, do the following:

1. Create index cards for the entries that you are considering for an area page.
2. Ask potential users to groups these index cards into piles that they feel go together.
3. Have users give a name to each pile.
4. Place the frequently accessed entities in the Common section. This section should provide navigation to a list page.

This technique can give you a good indication of how to organize the content on an area page or a group of area pages. For more information about card sorting, see "Card sorting: a definitive guide" at *http://www.boxesandarrows.com/view/card_sorting_a_definitive_guide*.

Avoid creating additional reports, inquires, and periodic forms for features related to a common entity. Instead, provide access to these forms through the entity's list page and details forms. This way, for example, the user doesn't have to search the area page for things related to a customer, but instead knows that all things related to a customer can be found on the Customer list page and details form.

Avoid adding multiple new pages to the Setup section. Instead, look for ways to consolidate setup information for a feature area into a single form by using the *table of contents* pattern. By consolidating this information, the user needs to find only one form and can easily see all related configuration information without having to go back to the area page.

Avoid creating new area pages that are specific to a custom solution unless this makes logical sense. For example, if your solution provides the capability to do credit checks on customers, it is typically better to add links to these features in the Accounts Receivable area than it is to create a new area page specifically for credit checks. Accounts Receivable users expect these features to be part of the Accounts Receivable area and not in a separate area.

Spend the time necessary to organize the content in a way that is logical to your users. Users will not only benefit while they learn to use your features, they will also benefit during extended use. Often, even experienced Microsoft Dynamics AX users struggle to remember where to find an infrequently used form. Typically, this happens because the form is accessed from a place that wasn't logical to the user.

List pages

List pages are the starting point for many tasks in Microsoft Dynamics AX. Any scenario that starts with finding a record or a set of records is best suited for a list page, as shown in Figure 5-7. List pages are designed to be the place where users can find information and then act on that information.

A simple scenario: taking a call from a customer

To fully understand the power of a list page, consider a simple scenario of a customer service representative (CSR) in a manufacturing company who receives calls from customers. When the CSR receives a call from a customer, she wants to be efficient and take the least amount of time while on the phone. This scenario is optimally suited for a list page. The steps in this scenario correspond to the numbered items in Figure 5-9 and illustrate how a CSR can use a list page to perform a group of related tasks without having to leave the list page.

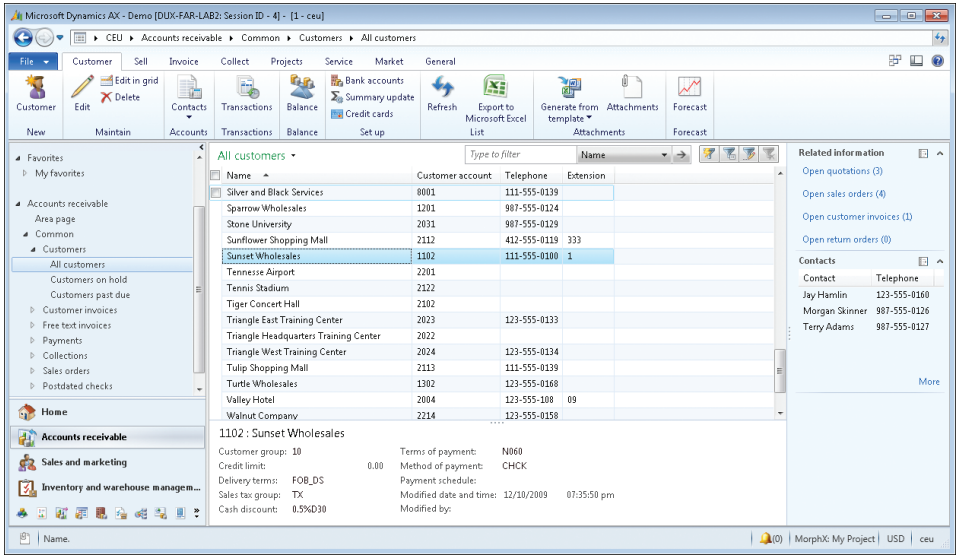


FIGURE 5-7 Customers list page.

1. When a call comes in, the CSR answers the phone while simultaneously opening the Customer list page. She assumes that a customer is calling.

2. The customer announces that his name is Terry and he is calling from Sunset Wholesales. The CSR greets the customer while typing **Sunset** into the Quick Find field.

3. The CSR notes that only one customer record with *Sunset* in the name is displayed in the list. To verify that she has found the correct customer record, she looks at the FactBox on the right. It shows three contacts from Sunset Wholesales, and Terry's name is in the list.

4. The customer wants a quote on purchasing 50 48-inch high-definition flat-screen televisions. The CSR clicks the Sales tab of the Action pane and clicks the new Sales Quotation button.

5. She proceeds to enter the quotation and quotes a price for the customer.

If, in this scenario, the customer Sunset Wholesale was not already in the system, the CSR would have searched for the customer, but no match would have been found. In this case, she could easily add a new customer from the Action pane, as shown in Figure 5-9.

If the customer called to check on the status of an existing order, the CSR could quickly check the Related Information FactBox (Figure 5-10) to see all the current open sales orders. By clicking this link, she is taken to the Sales Order List page, which displays the open orders for this customer.

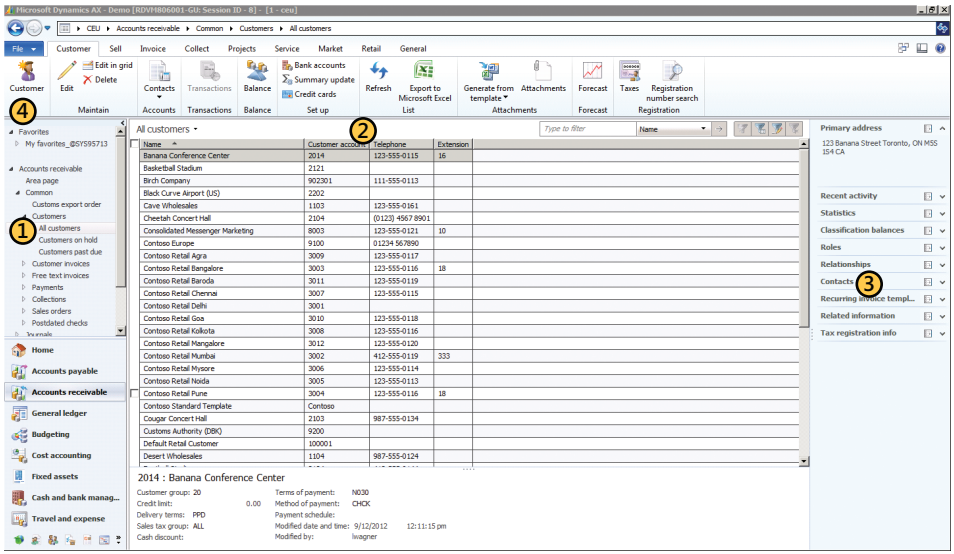


FIGURE 5-8 Steps to taking a call from a customer.

FIGURE 5-9 Adding a new customer.

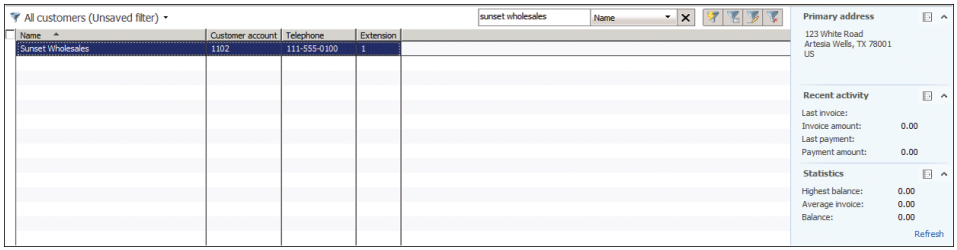


FIGURE 5-10 Customer-related information.

Use a list page as an alternative to a report

A list page is a great alternative to a traditional report. Historically, ERP systems have been focused on traditional reports as a way to get information out of the system. To an extent, Microsoft Dynamics AX 2012 has migrated away from traditional reports, and instead uses list pages as a place to view simple reports. For example, the customer aging list in Accounts Receivable > Collections is shown in Figure 5-11. This is a list of customers that displays information typically seen in an aged trial balance report. Having an interactive list of customers is much better than a traditional report because the user can sort this list easily by customer balance to see the customers who owe the organization the

most money at the top. Additional information about this customer is easy for the users to see in the FactBoxes. A user who wants to take action with this customer has full access to commands that are related to a customer. The Collections list page is also a great example of a role-tailored experience. This page displays a list of customers with specific information that Collections users need to see.

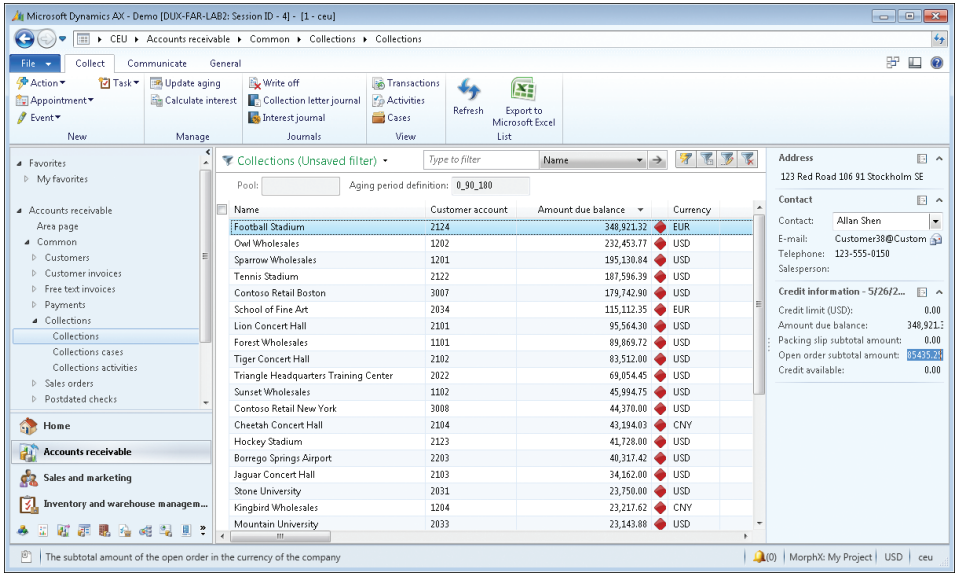


FIGURE 5-11 Collections list page displaying an aged trial balance report.

These are all examples that demonstrate how a list page is a great starting point for many scenarios. In these scenarios, it allows the user to find the customer and take the appropriate action quickly. Notice that when the CSR received the phone call, she did not know what the customer was calling about. Starting from the Customer list, she could easily find the customer, then wait for the customer to state what he or she wanted. At this point, she could take any action she needed to help the customer in a timely manner.

While this example focused on the Customer list, you'll see similar benefits in other list pages as well.

Design list pages

As a developer extending Microsoft Dynamics AX, you will need to extend an existing list page or design a new list page for an entity. Here are some tips to consider when you design a new list page or extend an existing one:

- Organize the tabs of the Action pane by activity. For example, on the Customer list page, we organized the commands based on typical activities that you perform against a customer, such as Sell, Invoice, Collect, etc. This helps the user find commands more easily, especially if there are many actions.

- Provide access to all actions that the user needs to perform against the entity in the Action pane. Users expect all actions to be available from the list page. Don't force them to go elsewhere to initiate an action.
- Allow the user to perform bulk actions by multiselecting items in the list. This is one of the most powerful capabilities of a list page because the user can easily filter the list and then select all records to take an action against.
- Provide secondary list pages that are filtered to show a specific set of records that need to be accessed frequently by the user. These secondary list pages should be added as a cue in the corresponding Role Center. This helps the users monitor the number of records in the list and get quick access to the list by clicking on the cue in the Role Center. Past Due Customers and Customers on Hold are examples of secondary lists that are included on the Customers list page.
- Design FactBoxes to display information that the user typically would have to open additional forms to see. By providing this information in a FactBox, you greatly simplify the user's experience because no additional action is required.
- Consider which columns the user needs to see in the list, and display those columns by default. Although the product provides users with a mechanism for adding columns to a list page, it is best if you can ensure that the fields the user needs to see are displayed automatically.
- Ensure that the page loads quickly. Users expect the list to appear within 2 to 5 seconds. This will require that you to optimize the queries used to load the list page. For more information about optimizing queries, see Chapter 13.
- When adding a new list page, follow the Microsoft Dynamics AX 2012 User Experience Guidelines on MSDN (*http://msdn.microsoft.com/en-us/library/gg886610.aspx*) to ensure that the list, Action pane, and FactBoxes are designed appropriately and match the rest of the application.

Details forms

Details forms are the primary method for creating and editing primary entities such as customers, vendors, workers, and products. A user opens a details form by double-clicking a record on a list page. By default, the details form for an existing entity opens in read-only mode. To modify the record, the user can click Edit to switch the form to edit mode.

All fields of a details form are grouped into FastTabs that the user can expand and collapse, as shown in Figure 5-12.

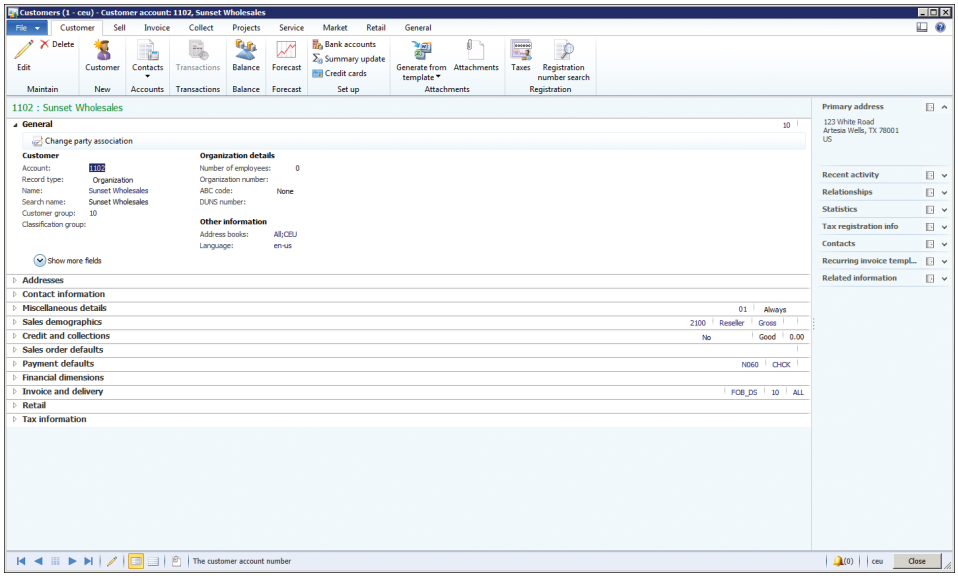


FIGURE 5-12 Customer details form.

FastTabs can display summary fields, which display key fields contained in the FastTab so that the user does not have to expand the FastTab. For example, in Figure 5-13, the summary field displays the customer's credit rating and payment terms, among other information.

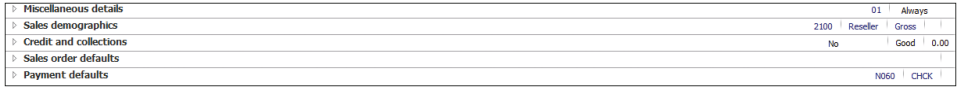


FIGURE 5-13 FastTabs with summary fields.

Details forms have an Action pane that display commands organized in the same way as the corresponding list page. The list page and the details form should have the same set of actions, with only a few exceptions. A details form also can contain FactBoxes to display related information. Many details forms contain the same set of FactBoxes as the list, but this is not a required feature. For more information, see the User Experience Guidelines (*http://msdn.microsoft.com/en-us/library/gg886610.aspx)*.

If you are introducing a new primary entity into Microsoft Dynamics AX 2012, you will need to create a new details form, in addition to a list page. Primary entities are typically tangible things that directly relate to the work a company performs, such as customers, vendors, employees, or inventory items. They tend to have many fields, many actions, and a great deal of related information.

The primary effort required for designing a new details form is to organize all of the fields within FastTabs. This exercise will require some knowledge of your users and the work they do with these entities. To organize fields into FastTabs, here are some guidelines to consider:

- Create FastTabs that are organized into groups that are logical to your users. This can be another situation where a card sort can help inform your decisions. Ask your users to organize

the fields of the entity into groups and then ask them to name the groups. As you go through this exercise, test the organization with your users to see if it is intuitive for them. It may take multiple iterations to organize the fields correctly. Don't be discouraged; multiple iterations are normal to get the correct design.

- Keep the number of fields in a FastTab as low as possible because taller FastTabs are less usable than shorter ones. When a tall FastTab is expanded, users lose their context in the form because a taller FastTab requires more scrolling and doesn't allow multiple FastTabs to be expanded at the same time.
- Order the FastTabs to put the most important FastTabs at the top and the least important ones at the bottom.

Here are a few other tips for designing a details form:

- Organize the tabs of the Action pane by activity. This helps the user more easily find their commands, especially if there are many actions.
- Provide access to all the actions that the user needs to perform against the entity in the Action pane. Users expect all actions to be available in the Action pane. Don't force users to go elsewhere to initiate an action.
- If multiple roles use the form, ensure that members of each role see only the commands that are required for their jobs. You can organize commands so that entire Action pane tabs are hidden from specific roles. You can configure this through the Microsoft Dynamics AX 2012 security model. For more information, see Chapter 11.
- Design FactBoxes to display information that the user would typically have to open additional forms to see. By providing this information in a FactBox, you greatly simplify the user's experience because no additional action is required.
- Ensure that the page loads quickly. The user will expect the form to open within 2 to 5 seconds. This will require that you to optimize the queries that are used to load the form. For more information about optimizing queries, see Chapter 13.
- Give users the capability to edit multiple records from within the details form, as shown in Figure 5-14. The user can initiate this through the Grid View button on the status bar.

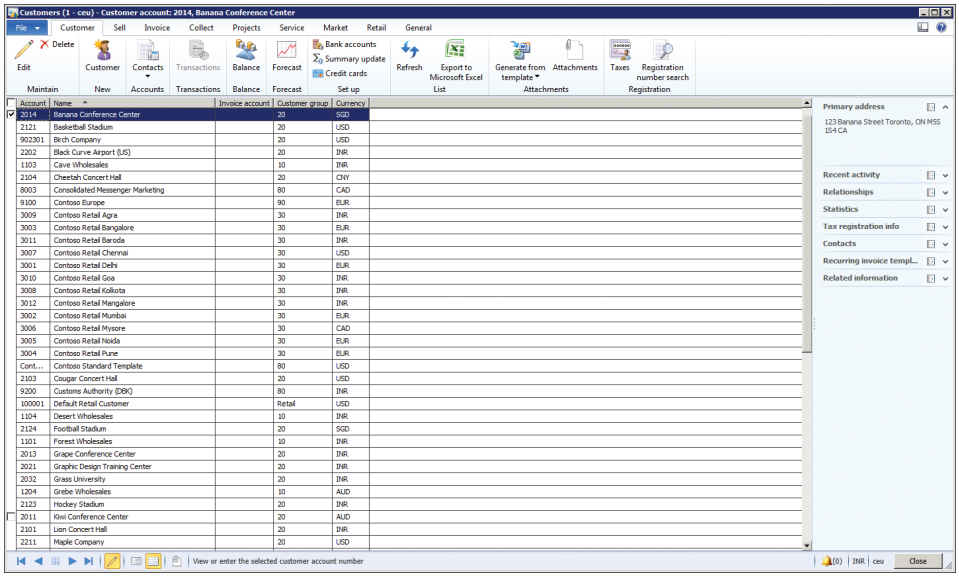


FIGURE 5-14 Grid view of the Customer details form.

Transaction details forms

Transaction details forms are used for creating and editing transactions in Microsoft Dynamics AX. A transaction is a business event that occurs within a company that needs to be recorded in the ERP system. Examples of transactions in Microsoft Dynamics AX are sales orders, purchase orders, invoices, bank deposits, and so on. The user experience for recording transactions is critical for any ERP system because many transactions must be recorded on a daily basis. Transaction details forms must be optimized for efficiency so that users can enter new transactions easily. These forms must be intuitive so that users don't make mistakes that cost time and money to resolve. Users of these forms typically use them repeatedly throughout the course of the day. They learn every nuance of the form to become as efficient as possible, and they become frustrated by any extra step that is required because, over the course of a day, the extra step slows them down.

The Sales Order and Purchase Order transaction details forms are possibly the most complex forms within Microsoft Dynamics AX because of the number of fields and actions that they need to support. With each release, new fields and actions are typically added to these forms to support additional capabilities that customers request. Figure 5-15 shows the Sales Order detail form, which is used to create new sales orders. This form has been simplified by providing quick access to the important header fields and the lines of the order. It has been optimized for the orders that are typically created by a user, while still supporting all possible options. This will require many users to customize these forms to meet their needs. The goal with these forms is to design a great experience that can be customized easily for the specific needs of each user.

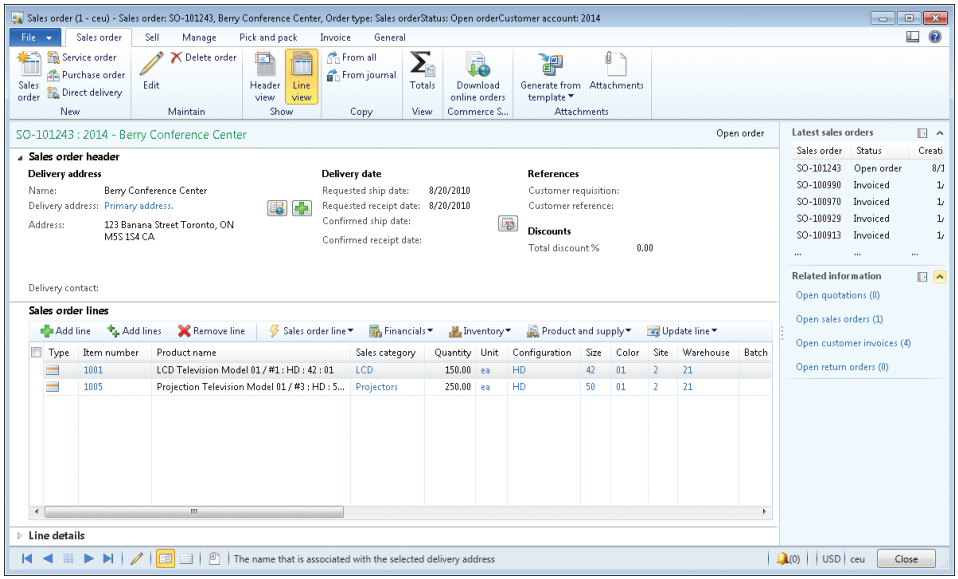


FIGURE 5-15 Sales Order details form.

Transaction details forms are similar to details forms because users open them from a list page by double-clicking a transaction record. By default, transaction details forms open in read-only mode the same way that a details form does. To modify the record, a user clicks Edit to switch the form to edit mode. Transaction details forms differ from details forms because they typically have line items to indicate the details of the transaction. The line items are the main focus of these forms and are where the users spend most of their time. Transactions can vary greatly in their complexity: a simple transaction might require only 1 or 2 line items, but a complex transaction might require more than 100 line items. Transaction details forms must be designed to accommodate both of these situations.

A transaction details form has two views, which users can toggle between by using buttons in the Action pane:

- **Line view** Displays only the header fields that are most likely to be needed when a user creates a new transaction. Line view is the default view and is designed to support the majority of the user's tasks. You should modify this set of header fields to display the most important fields for your users.
- **Header view** Displays all the header fields of the transaction. Typically, many of these fields use default values and are not completed directly by a user. These fields are omitted from the Line view to make it easier to use.

As a developer extending Microsoft Dynamics AX, you may need to extend an existing transaction details form or design a new transaction details form for an entity. Here are some guidelines to consider when you extend or design a new transaction details form:

- Organize the tabs of the Action pane by activity. This helps the user more easily find commands—especially if there are many actions.
- Provide access to all actions that the user needs to perform against the entity in the Action pane. Users expect all actions to be available in the Action pane. Don't force users to go elsewhere to initiate an action.
- If multiple roles use the form, ensure that members of each role see only the commands that are required for their jobs. You can organize commands so that entire Action pane tabs are hidden from specific roles. You can configure this through the Microsoft Dynamics AX 2012 security model. For more information, see Chapter 11.
- Ensure that the columns in the line items list are the fields that the user completes most frequently. Entering fields into the grid is much more efficient than using the line details at the bottom of the form.
- Design FactBoxes that display information to users that help them while entering new transactions or when viewing an existing transaction. By providing this information in a FactBox, you greatly simplify the user's experience because no additional action is required to see this information.
- Ensure that the page loads quickly. The user will expect the form to display within 2 to 5 seconds. Performance of this form is extremely critical because it is used repeatedly throughout the day. Any performance issue on the forms will frustrate the user.
- When adding a new transaction details form, follow the Microsoft Dynamics AX 2102 User Experience Guidelines at *http://msdn.microsoft.com/en-us/library/gg886610.aspx*. Note that the guidelines refer to transaction details forms as details forms with line items.

Enterprise Portal web client user experience

The Enterprise Portal web client provides a similar user experience to the Microsoft Dynamics AX Windows client. Any user who is familiar with the Microsoft Dynamics AX client should also feel comfortable using Enterprise Portal. Like the Microsoft Dynamics AX client, Enterprise Portal also contains navigation layer and work layer forms, but the navigation path is simplified, as shown in Figure 5-16.

The following sections describe the Enterprise Portal user experience. For more information about creating Enterprise Portal pages, see Chapter 7, "Enterprise Portal."

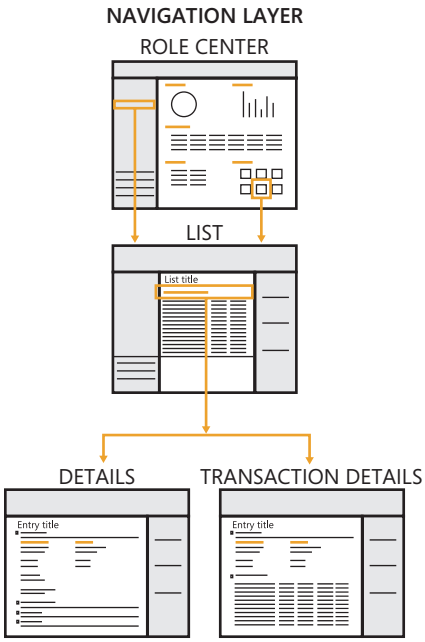


FIGURE 5-16 Enterprise Portal navigation paths.

Navigation layer forms

In Enterprise Portal, navigation layer forms include Role Centers and list pages. The user has the same Role Center between the Microsoft Dynamics AX client and Enterprise Portal. List pages in Enterprise Portal are similar to those in the Microsoft Dynamics AX client and, from a developer perspective, are actually the same form (see Chapter 7). All Enterprise Portal navigation layer forms appear within the Enterprise Portal workspace, which consists of the following components:

- **Top navigation bar** Contains a set of links at the top of the page. A user can use this bar to navigate between the various areas, such as Sales and Procurement, that are visible to him or her. Each link in the top navigation bar points to the default page of the corresponding area.
- **Search bar** Lets users search for help content and data and forms. By default, users can use the search bar to search for Microsoft Dynamics AX help and Microsoft SharePoint help. If you want to enable searching for data and forms, Enterprise Search must be configured as part of the setup. For more information, see "Enterprise Search" at *http://technet.microsoft.com/en-us/library/gg731850.aspx*.
- **Action pane** Displays a set of buttons that are categorized into contextual tabs and button groups similar to the Action pane in the Microsoft Dynamics AX client and Microsoft Office applications. This enhances simplicity and discoverability because the actions available vary based on the permissions of the user.

- **Navigation pane** Contains a set of links on the left side of the page that allow a user to navigate to the various areas and pages within an area. Note that the Navigation pane in Enterprise Portal doesn't provide access to all areas; instead, it provides navigation within an area. This differs from the Navigation pane in the client.
- **Content pane** Appears to the right of the Navigation pane. The Content pane displays content pages such as Role Centers, as well as list pages.
- **FactBox pane** Appears at the right of the workspace and provides related information about a specific record in a grid. The FactBox pane is displayed only on list pages within the workspace. Unlike FactBoxes in the Microsoft Dynamics AX client, the FactBox pane in Enterprise Portal cannot be personalized by the user.

Work layer forms

The primary work layer forms in Enterprise Portal are details forms, which are used for entering information into Microsoft Dynamics AX. A details form lets users view, edit, and act upon data. These forms are similar to the details forms in the Microsoft Dynamics AX client but have a smaller set of fields and actions.

Design for Enterprise Portal

When you are designing for Enterprise Portal, consider where users will perform similar actions in the Microsoft Dynamics AX client, and plan the user experience so that it is consistent:

- Organize the content into areas similar to those in the Microsoft Dynamics AX client. If users find customers in the Sales and Marketing area of the client, they will expect customers to be located in the same area of Enterprise Portal.
- Organize commands in the Action pane in a similar manner to those on the client.

For more information about designing new forms for Enterprise Portal, see the Microsoft Dynamics AX 2012 User Experience Guidelines at *http://msdn.microsoft.com/en-us/library/gg886610.aspx*.

Design for your users

This chapter has talked about how to design powerful and simple user experiences for your users. The key to designing powerful and simple experiences is to truly understand your users so that you can focus your designs on what the user is likely to do. Don't assume that you know what your users know or what they need or want. Also, don't assume that their managers know what they need or want. Instead, take the time to observe them working, and talk to them about what they need. Also, keep in mind that sometimes users cannot articulate what they need or want. You will have to develop the skills to observe and listen for their unarticulated needs.

Based on the insights you gain, sketch out some possible designs take them back to the users for their feedback. Avoid prototyping the solution; instead, simply create a sketch. This might feel

awkward if you think that you need to have a perfect design before you take it back to users. Keep in mind that they will appreciate the opportunity to provide feedback early in the process. When they see that you haven't invested a lot of time on your designs, they will be more willing to provide feedback. If you get feedback that you are off the mark on your designs, it is easy for you to change direction at this point because you haven't invested a lot of time on your sketches.

When you are getting feedback from your users on your designs, don't demo the design to them and ask for their opinion. Instead, ask them to explain what they are seeing with these designs and describe how they think they would take actions with them. If they are able to describe how the designs work and indicate how they can be used, you are on the right track. If not, take their input and sketch out some new designs. Don't go too long without talking to your users, and don't be afraid to fail. The key is to fail early when you haven't invested much time in your designs, and to determine the right design before you begin coding your feature. Create two or three iterations until you get a design that seems to resonate with users. Remember that designing a simple, easy-to-use feature is a difficult exercise.

CHAPTER 6

The Microsoft Dynamics AX client

In this chapter

Introduction 159
Working with forms 159
Adding controls 172
Using parts 181
Adding navigation items 182
Customizing forms with code 184
Integrating with the Microsoft Office client 189

Introduction

At its core, the Microsoft Dynamics AX Windows client is a form-based Windows application that lets users interact with the data contained on the server. You can modify the client to display new data types or to alter how users interact with existing data types. The user interface consists of forms that are declared in metadata and often contain associated code.

Microsoft Dynamics AX 2012 includes several updates and additions for the client. Some of the more substantial changes include new patterns for master records (details forms) and secondary data (list pages and transaction details forms), a new vertically expanding FastTabs control, the use of Action panes and Action pane strips to display actions more prominently, and the introduction of FactBoxes to showcase related information. For more information, see Chapter 5, "Designing the user experience."

The majority of this chapter covers key aspects of the Microsoft Dynamics AX client. However, some of the information in this chapter is at the overview level. For more detailed information, see the "Client" section of the Microsoft Dynamics AX 2012 SDK at *http://msdn.microsoft.com/en-us/library/gg880996*.

Working with forms

A form is the basic *unit of display* in the client. A typical form displays fields that show the current record, buttons that represent the actions the user can take on that record, and a mechanism to change which record is being shown.

To create a form, you use the Application Object Tree (AOT) to define the metadata for the form. If necessary, you can add code to handle any events that cannot be handled declaratively in metadata. The following high-level steps describe the basic process for creating a form:

1. Create the form resource. You can create a form from scratch, but often, you can use an existing form or a template as a starting point. When you create the form, be sure to set the *Caption* property on the form's *Design* node. This is an important but often overlooked step.
2. Add data sources and set up join information. You can define custom queries and filters, if necessary.
3. Add controls to the form. You can add controls that are bound to fields to display data and action controls such as buttons and Action panes that let the user perform actions on the current record.
4. Add parts to the form that display data related to the main record. Parts can reduce the navigation that users must perform to find information.
5. Add navigation items so that users can access your form. Create a *MenuItem* control that points to the form. Add a reference to that *MenuItem* to *Menu* controls, or to other forms through *MenuItemButton* controls, to let the user to navigate to the form.
6. Override form and control methods if you cannot achieve the behavior that you want declaratively through metadata.
7. Add business logic to classes as necessary to implement the functionality that the new form provides.

The following sections in this chapter contain more information about the components in each step. For the latest information and most up-to-date examples about how to build and customize forms, see the "Client" section in the Microsoft Dynamics AX SDK at *http://msdn.microsoft.com/en-us/library/gg880996*.

Form patterns

Earlier releases of Microsoft Dynamics AX had informal patterns for form development. In Microsoft Dynamics AX 2012, several form patterns have been formalized and are provided as templates.

When you create a form, select a form pattern that reflects the type of data that appears in the form and the interaction pattern that is provided to the user. The "Form User Experience Guidelines" on MSDN (*http://msdn.microsoft.com/EN-US/library/gg886605*) discuss each of the form patterns. These guidelines are useful to ensure a seamless experience between the new form and the existing forms in Microsoft Dynamics AX.

After you select a form pattern, you can create a form by using a template:

- In the AOT, right-click the *Forms* node, click New Form from Template, and then select the template you want.

Microsoft Dynamics AX generates from the template that contains property values and controls that implement the structure specified by the form pattern. Table 6-1 describes the form templates that are available and the purpose of each type of form.

TABLE 6-1 Form templates.

Pattern	Template	Purpose
List page	ListPage	Find a record and perform an action on it. A separate details form is used to show the details about that record. This pattern is intended for primary or master records. Example: CustTableListPage To open this form:. Under Accounts Receivable, click Common > Customers > All Customers.
Details form	DetailsFormMaster	View, enter, update, and perform other actions on an individual record. This pattern is intended for primary or master records. Example: CustTable. To open this form: Under Accounts Receivable, click Common > Customers > All Customers, and then double-click an entry in the list.
Details form with lines	DetailsFormTransaction	View, enter, update, and perform other actions on an individual record that is associated with one or more related lines. In addition, the form enables you to perform actions on that record and its lines. Example: SalesTable To open this form: Under Accounts Receivable, click Common > Sales Orders > All Sales Orders, and then double-click an entry in the list.
Dialog	Dialog	Initiate a task or process where the user must provide input. The form lets users specify whether to continue or cancel the task or process. Example: DirPartyQuickCreateForm To open this form: Under Accounts Receivable, click Common > Customers > All Customers. On the Action pane, in the New group, click Customer.
Drop dialog	DropDialog	Initiate a task or process where the user must provide input. A drop dialog provides a small amount of information quickly without requiring the user to leave the parent form. Example: HcmWorkerNewWorker To open this form: Under Human Resources, click Common > Workers > Workers. On the Action pane, in the New group, click Hire New Worker.

Pattern	Template	Purpose
Simple list	SimpleList	View, enter, and update records that appear as a list of records in a grid. Example: CustGroup. To open this form: Under Accounts Receivable, click Setup > Customers > Customer Groups.
Simple list and details	SimpleListDetails	View a list of records and the details about one of those records at the same time. This pattern is targeted at simpler secondary records. Example: CustPosting To open this form: Under Accounts Receivable, click Setup > Customer Posting Profiles.
Table of contents	TableOfContents	Complete a series of related setup or configuration tasks. The table of contents pattern is used on parameters forms to allow easy access to the parameters that the current module is using. Example: CustParameters To open this form: Under Accounts Receivable, click Setup > Accounts Receivable Parameters.

Note There are no formalized patterns for journal and inquiry forms because the structure of those forms are highly dependent on the data and processes they support.

Form metadata

The form metadata in Microsoft Dynamics AX is extensive, but it is well-structured and easy to work with after you become familiar with it. The following are the primary metadata nodes for a form resource:

- ***Form.DataSources*** The data structures that are used for the form. For more information, see "Form data sources," later in this chapter.
- ***Form.Designs.Design*** The controls that display the data for the record. This metadata node name is often shortened to *Form.Design* or *Form Design*. For more information, see "Adding controls," later in this chapter.
- ***Form.Parts*** The additional parts that display related data. For more information, see "Using parts," later this chapter.

Figure 6-1 illustrates these nodes in the AOT for the CustGroup form.

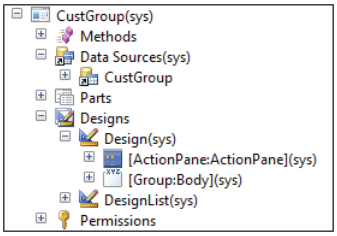


FIGURE 6-1 Metadata nodes for the CustGroup form.

Ideally, you should use metadata to customize forms. Metadata customization is preferred over code customization because metadata changes (also called deltas) are easier to merge than code changes. To ensure the greatest level of reuse, any changes you make to the metadata should be made at the lowest level possible; for example, at the table level instead of the form level.

When customizing forms, you should be aware of the metadata associations and the metadata inheritance that are used to fully define the form and its contents.

Metadata associations

You edit the metadata in Microsoft Dynamics AX by using the AOT. The base definitions for forms contained within the *AOT\Forms* node consists of a hierarchy of metadata that is located in other nodes in the AOT. To fully understand a form, you should investigate the metadata associations it makes. For example, a form uses tables that are declared in the *AOT\Data Dictionary\Tables* node, menu items that are declared in the *AOT\Menu Items* node, queries that are declared in the *AOT\Queries* node, and classes that are declared in the *AOT\Classes* node.

Metadata inheritance

You need to be aware of the inheritance within the metadata used by forms. For example, tables use base enums, extended data types (EDTs), and configuration keys. A simple example of inheritance is that the *Image* properties on a *MenuItemButton* are inherited from the associated *MenuItem* if they aren't explicitly specified on that *MenuItemButton*.

Inheritance also occurs within forms. Controls that are contained within other controls receive certain metadata property behaviors from their parents unless different property values are specified, including Labels, HelpText, Configuration Key, Enabled, and the various Font properties.

Table 6-2 shows examples of pieces of metadata that are inherited from associated metadata.

TABLE 6-2 Examples of metadata inheritance.

Type of metadata	Sources
Labels and help text	MenuItem > MenuItemButton Control Base Enum> Extended Data TypeTable Field> Form DataSource FieldForm Control (The *base enum Help* property is the equivalent of the *HelpText* property found in the other types.)
Display length	Extended Data Type > Table Field > Form Control
Configuration keys	Base Enum > Extended Data Type > Table Field > Form DataSource Field > Form Control
Image properties (for example *NormalImage*)	MenuItem > MenuItemButton Control

Form data sources

Microsoft Dynamics AX has a rich data access framework that makes it easy to add data to forms and bind controls to that data. The basis of this is the form data source, which allows binding the tables and fields to a form.

The form data source points to a specific table, map, or view. The field list on the form data source is automatically populated with the fields that are defined on the resource it refers to. From that list, you can bind controls to those fields or any of the data methods that exist on the table or form data source.

Form data sources can be divided into the following categories:

- **Root data sources** Root data sources do not contain a value for the *JoinSource* property and therefore, are not joined with or linked to any other data source. Most forms have only one root data source. The root data source references the table data that is the primary subject of the form. Root data sources are sometimes called *top-level* data sources.
- **Master data sources** Master data sources are root data sources or dynalinked data sources. A single query is used to retrieve the data for a master data source and the data sources that are joined to it. You can think of a master data source as being at the root of a query hierarchy.
- **Joined data sources** Joined data sources are those that are joined to another data source. These data sources have a *LinkType* value of *InnerJoin*, *OuterJoin*, *ExistJoin*, or *NotExistJoin*. Typically, you use a join to combine data sources so that the data is retrieved by a single query. For example, in the CustTable form, DirPartyTable is joined to CustTable.
- **Linked data sources** Linked data sources are data sources that are linked to another data source in the form. These data sources have a *LinkType* value of *Active*, *Delayed*, or *Passive*. Use a link for data sources that have a parent/child relationship so that the data is retrieved in separate queries. For example, in the SalesTable form, SalesLine is linked to SalesTable.

Dynalinks

The term *dynalink* refers to two data sources that are dynamically linked. A dynalink always has a parent data source and a child data source.

For example, the SalesTable (Sales orders) form where the SalesTable (Sales order) data source is the parent and the SalesLine (Sales order line) data source is the child. If two data sources have a dynalink, when a record changes in the parent data source, the child data source is notified about that change. The query for the child data source is reexecuted to retrieve the appropriate related data.

The following types of dynalinks are available:

- **Intra-form** Intra-form dynalinks occur between data sources that have a *LinkType* value of *Active*, *Passive*, or *Delayed*. The child data source has a query that is separate from the parent data source, and the query runs at a time that is determined by the *LinkType* property.
- **Inter-form** Inter-form dynalinks occur between related data sources on forms where one form (the parent) opens another form (the child). The *DataSource* property of a *MenuItemButton* on the parent form is used to specify which data source is used as the parent form side of the link. The child form side of the link is the first root data source.

Table inheritance

Table inheritance in Microsoft Dynamics AX functions much like class inheritance in any object-orientated language. However, it has the added benefit of allowing polymorphic queries of the data in tables. (For more information about table inheritance, see Chapter 17, "The database layer.")

If you model a form data source on a table that is a base type, the derived types are automatically expanded into a subnode called *Derived Data Sources*. This node is not editable and is generated by the forms engine. The derived data sources have no properties or methods of their own because all of those characteristics are inherited from the base form data source. However, you can still override and add methods to the fields for derived data sources. For example, the DirPartyTable data source, shown in Figure 6-2, is part of a table inheritance hierarchy.

Figure 6-2 shows all of the automatically-generated data sources, one for each derived type. The *Fields* node for each derived type lists the fields for that type. When there is only a single chain of base types, all of the base fields are collapsed into a single *Fields* node underneath the form data source. For example, if you model a form data source based on the DirPerson type, the *Fields* node contains all of the fields in the chain.

Instance methods on the form data source also follow the table inheritance hierarchy. For example, if the user triggers the *validateWrite* event when the current active record is of type DirPerson, the *FormDataSource.validateWrite* method is called, which will call *DirPerson.validateWrite*, which will call *DirPartyTable.validateWrite*, which will call the kernel level *Common.validateWrite*. However non-instance-specific methods such as *executeQuery* work on the general form data source, so there are no calls to any base methods.

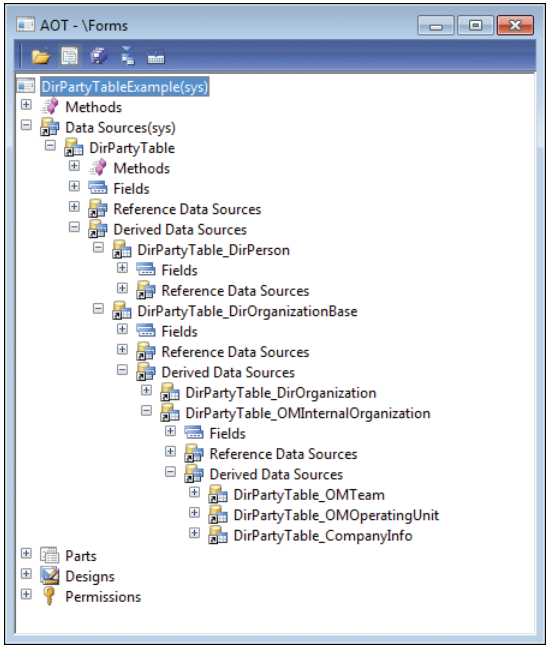


FIGURE 6-2 The DirPartyTable data source in the AOT.

Because polymorphic queries are allowed, polymorphic creation of records is also supported. When a user clicks New on a form with a form data source that has no derived types, the concrete type to create is known. However, when the form data source has derived types, the user must be prompted to select a type to create.

Traditionally, to create a record in X++ code you only had to call the *FormDataSource.create* method. However, that method does not let you specify the type. To support the polymorphic creation scenario, use the following method:

```
FormRun.createRecord(str _formDataSourceName [, boolean _append = false])
```

All create actions performed by the kernel are routed through this method. You should use this method as well instead of the *create* method. The first parameter specifies the name of the form data source in which to create the record, and the second parameter contains the same *append* value that is passed to the *create* method. You can override this to put in conditional code that depends on the type being created. The call to the *super* of the method executes the correct logic depending on the type.

If the type (or any of the types in the join hierarchy) is a polymorphic type, the user is prompted to select the type of record to create, as shown in Figure 6-3.

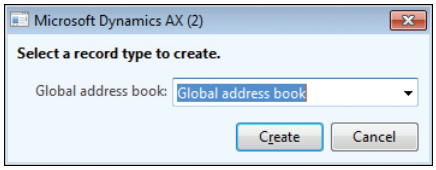


FIGURE 6-3 Dialog box that prompts a user to specify a record type.

The *createRecord* method lets you override the behavior of the *super* to either specify the types of records that the user can create, or to display your own dialog box to the user. To inform the kernel of which types you want the user to choose from, you use the following method:

```
FormDataSource.createTypes(Map _concreteTypesToCreate [, boolean _append = false])
```

The first parameter contains a map of *string* key/value pairs where the key is the name of the form data source and the value is the name of the table type to create. For example, if your objective is to always create a type of CompanyInfo, you could use the following code. Note that the name of the form data source is the name of the data source that is modeled on the form, not the derived data source.

```
public void createRecord(str _formDataSourceName, boolean _append = false)
{
    Map typestoCreate = new Map(Types::String, Types::String);

    if(_formDataSourceName == "DirPartyTable")
    {
        typestoCreate.insert("DirPartyTable", "CompanyInfo");
        DirPartyTable_ds.createTypes(typestoCreate, _append);
    }
    else
    {
        super(_formDataSourceName, _append);
    }
}
```

Unit of Work

Saving records in form data sources can occur in two ways. The traditional approach is that all of the inner-joined and outer-joined data sources are saved together in one process, but each record is saved to the server in individual remote procedure calls (RPCs) and transactions. This is can be troublesome, because sometimes you need all of the records to be saved in a single transaction. To achieve this, you can set the *ChangeGroupMode* property on the *Data Sources* node to *ImplicitInnerOuter*. (The default setting is *None*, which results in the behavior described earlier.) With the *ImplicitInnerOuter* setting, all of the inner-joined and outer-joined records are grouped into a single RPC to the server and occur in a single transaction. If anything causes the transaction to be cancelled, the changes are rolled back. This feature is called Unit of Work. For more information about Unit of Work, see Chapter 17.

With this new approach, the *write* and *delete* methods no longer apply, because the actions occurred in the call to the *super* for those methods. With the change group mode behavior, the *writing*, *written*, *deleting*, and *deleted* methods are used. For each type of operation, when the validate method for each data source is called on the client, the methods ending in *ing*, such as *writing*, are called. The transaction then occurs on the server, where the table *insert*, *update*, or *delete* methods are called. Finally, the methods ending in *ed* or *en*, such as *deleted* or *written*, are called.

Unit of Work has an additional feature called *OptionalRecord* that saves database space by inserting outer-joined records only if the values have been changed from the default. For *OptionalRecord*, the two possible options are *ImplicitCreate* and *ExplicitCreate*. In the *ImplicitCreate* scenario, the forms engine automatically creates the outer-joined record if the record does not exist in the database. The record is saved only if values are changed from the default value. In the *ExplicitCreate* scenario, you can model a check box on the form that explicitly controls the behavior of record creation and deletion for the outer-joined record.

Date effectivity

Date effectivity allows tracking of how data changes over time. The date effectivity functionality lets users with appropriate security privileges see the entire change history for a record. It also provides support for creating records that become effective on specific dates. Interest and currency conversion rates are good examples of date effective records as they make use of specific effective dates and times. For more information, see Chapter 17 or download the white paper, "Using date effective data patterns" from *http://download.microsoft.com/download/4/E/3/4E36B655-568E-4D4A-B161-152B28BAAF30/Using_Date_Effective_Patterns_AX2012.pdf*.

Surrogate foreign keys

Traditional foreign keys in Microsoft Dynamics AX used the natural key or a type of intelligent key for the target object. This has many drawbacks, such as breaking referential integrity if the value of a natural key was changed. To solve those problems and improve performance through the use of integer keys, surrogate foreign key support was added to Microsoft Dynamics AX 2012. Surrogate foreign key support uses reference data sources and Reference Group controls to provide surrogate key support in the Microsoft Dynamics AX client. For more information, see the Chapter 17.

Metadata for form data sources

Table 6-3 describes some of the most important form data source properties.

TABLE 6-3 Metadata properties for form data sources.

Property	Description
Name	Specifies a named reference for the data source. A best practice is to use the same name as the table name.
Table	Specifies the table used as the data source.
CrossCompanyAutoQuery	Specifies whether the data source gets data from all companies. The following values are available: ■ ***No*** (Default) Data source gets data from the current company. ■ ***Yes*** Data source gets data from all companies within the current partition; for example, the data source retrieves customers from all companies).
JoinSource	Specifies the data source to link to or join to as part of the query. For example, in the SalesTable form, SalesLine is linked to the SalesTable data source. Joined data sources are represented in a single query whereas a linked data source is represented in a separate query.
LinkType	Specifies the link or join type used between this data source and the data source specified in the *JoinSource* property. Joins are required when two data sources are displayed in the same grid. Joined data sources are represented in a single query whereas a linked data source is represented in a separate query. **Dynalinks** The following values are available: ■ ***Delayed*** (Default) A delay is inserted before linked child data sources are updated, enabling faster navigation in the parent data source because the records from the child data sources are not updated immediately. For example, the user could be scrolling past several orders without immediately seeing each order line. ■ ***Active*** The child data source is updated immediately when a new record in the parent data source is selected. Continuous updates consume significant resources. ■ ***Passive*** Linked child data sources are not updated automatically. The link is established by the kernel, but you must trigger the query to occur by calling *executeQuery* on the linked data source. **Joins** The following values are available: ■ ***InnerJoin*** Selects records from the main table that have matching records in the joined table, and vice versa. There is one record for each match. Records without related records in the other data source are eliminated from the result. ■ ***OuterJoin*** Selects records from the main table whether or not they have matching records in the joined table. An outer join doesn't require each record in the two joined tables to have a matching record. ■ ***ExistJoin*** Selects a record from the main table if there is a matching record in the joined table. ■ ***NotExistJoin*** Selects records from the main table that don't have a match in the joined table.
InsertIfEmpty	The following values are available: ■ ***Yes*** (Default) A record is automatically created if none exists. ■ ***No*** The user must create the first record manually. This setting is typically used when a special record creation process or interface is used.

Property	Description
AutoSearch	The following values are available: ■ ***Yes*** (Default) *executeQuery* is called automatically during the call to the super of *FormRun.run*. ■ ***No*** *executeQuery* is not called during the call to *FormRun.run*. *This property is valid only on root data sources*.
AutoQuery	The following values are available: ■ ***Yes*** (Default) A query is automatically created by the FormRun engine. The query contains a *QueryBuildDataSource* object for every *FormDataSource* object in the join hierarchy. ■ ***No*** The FormRun engine does not automatically create a query. You must provide one by setting the *FormDataSource.query* object during the call to the *FormDataSource.init* method. *This property is valid only on master data sources*.
OnlyFetchActive	The following values are available: ■ ***No*** (Default) All of the fields for the *FormDataSource* are selected in the *QueryBuildDataSource*. Equivalent to setting the *Dynamic property* to *Yes* on the *QueryBuildDataSource.FieldList* object. ■ ***Yes*** Only fields that are bound to controls are added to the select list on the *QueryBuildDataSource*.

Form queries

One of the most common ways of customizing a form is to modify the queries that the form uses. There are two primary ways to do this: by using the *AutoQuery* property or by using an explicit query. When the form loads, the following processing occurs for form data sources:

1. Form data source objects are created that reference the data that is retrieved from the database.
2. The form's queries are run to retrieve data.
3. Controls that are bound to fields show data that was retrieved.

By default, when the *FormDataSource.AutoQuery* property is set to *Yes*, a query is created for the form based on its data sources. The query is created in the call to the *super* of the form's *init* method. The query contains a *QueryBuildDataSource* object for each form data source in the direct join hierarchy. Additional queries are created for dynalinked form data sources. These queries are linked to the current data in their joined parent, so that the queries are correct. This has the benefit of splitting tables across multiple database queries, which can improve performance and remove the cross-product results that occur in 1:*n* joins. Figure 6-4 shows the data sources for an example *AutoQuery*.

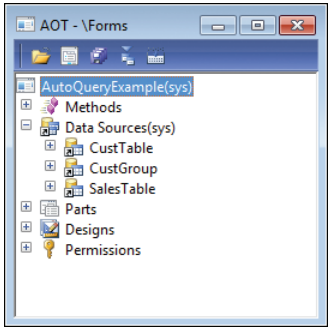


FIGURE 6-4 The AutoQueryExample form in the AOT.

In this example, properties are set on the data sources as follows:

- The CustTable data source is the root form data source; therefore, its *JoinSource* property is not set.
- The CustGroup data source has its *JoinSource* property set to CustTable with a *LinkType* value of *OuterJoin*.
- And the SalesTable data source has a *JoinSource* of CustTable and a value *LinkType* of *Delayed*.

With *AutoQuery* behavior, a query will be created with a root *QueryBuildDataSource* object of CustTable that has a child data source of CustGroup. Because the SalesTable data source is linked with a dynalink, it has its own query with a single root *QueryBuildDataSource* object. The SaleTable *QueryBuildDataSource* object has a dynalink added to it through the *addDynalink* method to the CustTable data course. When the form loads, the initial query runs to retrieve the data from CustTable and CustGroup. After the results are returned, a query runs to retrieve the SalesTable data based on the record in CustTable that is currently selected.

The second way of modeling the data access for a form is to use a query as the basis for the data source structure. You can do this by performing a drag-and-drop operation to add the query to the *Data Sources* node or by setting the *Query* property. In this scenario, the query causes the respective form data sources to be generated for the form. No additional data sources can be added. This method of modeling data access is generally used only for list pages, because it requires two metadata items to be created to model the form. The extra work is beneficial in the case of list pages, because composite queries are created that contain filters for the secondary list pages that reuse the same list page form. For example, Customers On Hold, which is a secondary list page, reuses the Customers list page.

QueryBuildDataSource and *QueryRunQueryBuildDataSource* methods

An important change in Microsoft Dynamics AX 2012 is the addition of the *FormDataSource.queryBuildDataSource* method and the *FormDataSource.queryRunQueryBuildDataSource* methods. These methods expose the respective *Query* and *QueryBuildDataSource* objects that the forms engine uses. These methods let X++ developers quickly access the correct *QueryBuildDataSource* object for the form data source. This is especially helpful when a form has multiple form data sources of the same type, such as the CustTable form.

Query and *QueryRun* objects

An important part of query interaction with the form is accessing the proper query to work with. For every data source on a form, there are two queries: the *FormDataSource.query* and the *FormDataSource.queryRun.query*. When a form initially loads, the *Query* is created in the call to the *super* of the *FormDataSource.init* method. Any modifications that you want to remain regardless of the filters that a user defines or clears should be applied to this query. In the call to the *super* of the *executeQuery* method, the *QueryRun* object is created, which contains a copy of the original *Query* object. When a user applies a filter, the filter is applied to *QueryRun.query* and the *research* method is called. An internal flag is set that specifies not to recreate the *QueryRun* object, and then the *executeQuery* method is called. When the user clears the filters, *executeQuery* is called, which by default, re-creates the *QueryRun* object from the base *Query* object.

CopyCallerQuery property

CopyCallerQuery is a property for the *MenuItemButton* and *MenuItem* resources that specifies whether to copy the query from the source form to the target form. When using *CopyCallerQuery*, the same rules apply as when you manually assign a query through code:

- The root data sources of the query must match the form data source.
- The joined data sources for the query should also be compatible.

In Microsoft Dynamics AX 2012, the kernel automatically adds any missing *QueryBuildDataSource* objects for required form data sources. This makes the queries as compatible as possible.

Query filters

Forms in Microsoft Dynamics AX 2012 apply filtering through the use of *QueryFilter* objects instead of *QueryBuildRange* objects. *QueryBuildRanges* are applied to the *ON* clause in a Transact-SQL statement, which works correctly for inner joins because the *ON* clause and the *WHERE* clause provide equivalent behavior. However, this does not work as expected for outer joins. The expected behavior for data sources with outer joins is to apply the *WHERE* clause to restrict the entire query, instead of the *ON* clause, which restricts only the join. To solve this problem, the *QueryFilter* class was created. All of the internal kernel logic in the forms engine that modifies queries uses *QueryFilter* objects. It is recommended that all new X++ logic on forms use *QueryFilter* objects instead of *QueryBuildRange* objects for consistency.

For more information about query filters, see Chapter 17.

Adding controls

Microsoft Dynamics AX includes a large selection of controls that you can use to create data-driven forms quickly.

When you add controls to a form, set as few properties as possible so that the controls can take full advantage of defaults and automatic values. Default and *Auto* property values on controls allow Microsoft Dynamics AX to use predefined functionality when determining display characteristics and behavior.

Note The product is an excellent source of information about how to build a form. You can see how forms in the product are built and which controls are used.

Control overrides

Each control has a set of methods that you can override. Try to keep code in the overridden method (on the form) to a minimum by calling a class instance or a static method when possible.

Control data binding

Many controls can be bound explicitly to a data source and data field to display the data field value. Other controls, such as buttons, can be bound to a data source to obtain the data context. If a control is not explicitly bound to a data source, its data source context comes from either its parent hierarchy or the form default, if the data source is not specified by a parent of the control.

The implied context of a data source is particularly important when actions are executed and records are saved:

- When an action executes, it executes in the context of a particular data source. For example, when it initiates an action to create a new record, the record that is created depends on the current data context.
- If changes to a record have not been saved when the cursor focus moves from a control in one data source context to a control in a different data source context, the record is saved automatically.

Design node properties

The *Design* node of a form contains the controls that display record data. The *Design* node contains properties, the most important of which are described in Table 6-4.

TABLE 6-4 Metadata properties for the *Design* node.

Property	Description
Caption	Specifies the caption text shown in the title bar of a standard form or in the filter pane of a list page.
TitleDataSource	Specifies the data source information displayed in the caption text of standard form and used to provide filter information in the caption text of a list page.
WindowType	Specifies the type of form. The following types are available: ■ ***Standard*** (Default) A standard single document interface (SDI) form that opens as a separate window with a separate entry in the Windows taskbar. This is the default type ■ ***ContentPage*** A form that fills the workspace content area. ■ ***ListPage*** A special type of *ContentPage* that displays records in a simple manner, providing quick access to filtering capabilities and actions. This type of form requires an Action pane control and a Grid control, at minimum. ■ ***Workspace*** A form that opens as a multiple document interface (MDI) window within the workspace. Workspace forms should be for developers only. ■ ***Popup*** A form that opens as a subform to its parent. Popup forms don't have a separate entry in the Windows taskbar and can't be layered with other windows.
AllowFormCompanyChange	Specifies whether the form allows company changes when used as a child form with a cross-company dynalink. The following settings are available: ■ ***No*** (Default) The form closes if the parent form changes its company scope. ■ ***Yes*** The form dynamically changes company scope as needed.

Runtime modifications

You can add or remove controls at runtime through code in response to user actions. For example, filter controls can be added and removed as needed.

When changing the form at runtime, you should lock it by using the *element.lock* and *element.unlock* methods to ensure the changes occur all at once instead of flickering into effect gradually. Locking the form can also improve performance because the the form is redrawn only once.

Action controls

Add action controls such as buttons and Action panes to let users perform actions on the current record.

Buttons

Users click buttons to perform actions. Several types of button controls are available:

- ***MenuItemButton*** Activates a *MenuItem* to open a form, run a class, or display a report. This is the most common type of button because the behavior is modeled rather than defined explicitly through X++ code. For more information, see "Adding navigation items," later in this chapter.

- ***CommandButton*** Executes a system-defined command, such as OK, Export to Excel, and Close. Use this type of button whenever a system-defined command exists for the action that you want to add.
- ***Button*** Provides a *clicked* method that you can override to add X++ code. This type of button is used infrequently. Avoid using it, if possible, because it means that code will be added directly to the form.
- ***DropDialogButton*** Opens a drop dialog form. Drop dialogs let the user quickly provide information or make choices that are necessary to execute some action. For example, with a drop dialog, a user can select a specific hold state when putting a customer on hold.
- ***MenuButton*** Displays a menu. This type of button can contain any button type except another *MenuButton*.

You should populate the *Text* property for a *Button* or a *MenuButton*. However, the text for a *CommandButton* or a *MenuItemButton* should come implicitly from the referenced command or *MenuItem*, respectively.

You can place buttons directly on a form or within a *ButtonGroup* that is on a form. More commonly, buttons are placed inside an Action pane or an Action pane strip.

Action pane and Action pane strip

An Action pane (see Figure 6-5) organizes and displays buttons that represent the actions the form supports. An action is a task or operation that occurs when the user clicks a button on the Action pane. With actions, you can use data that is displayed in the current form to let users perform commands, open related forms, or execute custom X++ code.

Use an Action pane at the top of large forms when you need multiple tabs to display the actions that are available for the entire form.

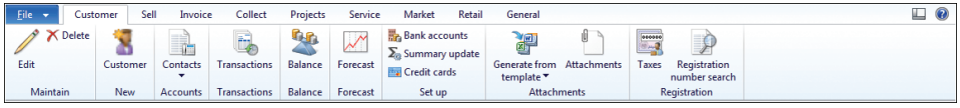


FIGURE 6-5 An Action pane.

Use an Action pane strip at the top of smaller forms when there are a small number of actions for the form. You can define an Action pane strip by setting the *ActionPane.Style* property to *Strip*.

You can also use an Action pane strip to display actions that have a specific context. Common locations for an Action pane strip are at the top of a TabPage, FastTab, or Group control. The context might be fields of a particular category within a record or a collection of associated child records. The most common usage of a contextual Action pane strip is when displaying the Add and Remove actions above a grid that contains associated child records, as shown in Figure 6-6. When you are using an Action pane strip in a particular data context, you should set the *DataSource* property of the Action pane control.

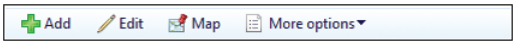


FIGURE 6-6 An Action pane strip.

For more information about how to design Action panes, see Chapter 5.

Layout controls

Three main layout controls are used to display other controls inside them:

- ***Group*** A *Group* control provides a way to group and categorize individual controls within the form. The *Design* node of a form, in addition to containing all of the controls, has many of the properties and behaviors of *Group* controls.
- ***TabPage*** A *TabPage* control organizes the controls and fields on the form so that only a subset is displayed at one time.
- ***Grid*** A *Grid* control displays input controls in a simple row and column format that allows the compact display of multiple fields for multiple records of the same type.

Group

Use the *Group* control to organized related fields and other controls into logical groups within a form. You can create and label a *Group* control manually (using the *Caption* property). You can also create a *Group* control by using the *DataGroup* property to point to a field group that has been predefined on a table (the data source).

Try to use table field groups whenever possible. Table field groups allow for easier maintenance of the application because a change to a table field group affects every form or report that uses that field group.

If you are manually adding controls and a caption to a *Group* control, be sure to provide a descriptive and understandable caption that accurately describes the group.

TabPage

Use *TabPage* controls to reduce the complexity of a form by hiding fields until the user needs them. *TabPage* controls are listed within a parent *Tab* control. The *Tab* control is commonly called a *tab group* to differentiate it from the *TabPage* controls it contains.

The following styles are available for *TabPage* controls:

- ***Standard*** Shows *TabPage* controls stacked on top of each other so that only one *TabPage* is visible at a time. This style, shown in Figure 6-7, is used throughout the product and should be familiar to most users.

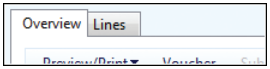


FIGURE 6-7 Standard tabs.

- ***VerticalTabs*** Shows *TabPage* controls listed as a vertically-organized set of links to the left of the *TabPage* that's visible, so that only one *TabPage* is visible at a time. This style, shown in Figure 6-8, is commonly used on parameters forms. Forms using this style are often said to have a *table of contents* style.

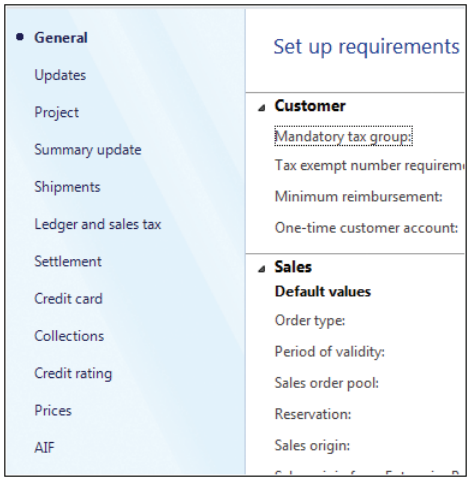


FIGURE 6-8 Vertical tabs.

- ***IndexTabs*** Shows *TabPage* controls listed as a horizontally-organized set of tabs underneath the *TabPage* that's visible, so that only one *TabPage* is visible at a time. This style, shown in Figure 6-9, is commonly used to display the line details on transaction detail forms.

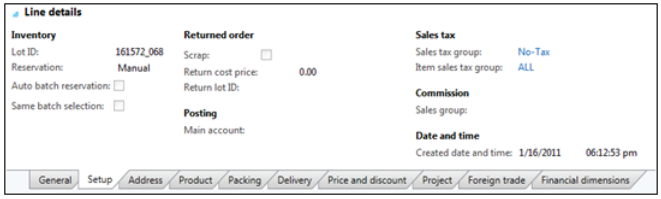


FIGURE 6-9 Index tabs.

- ***FastTabs*** Shows *TabPage* controls listed vertically and lets users show multiple *TabPage* controls at a time. The user can choose which pages to see and expand and collapse *TabPage* controls as necessary. With the *FastTabs* style, shown in Figure 6-10, you can also display summary information for key fields, even when the control is collapsed.

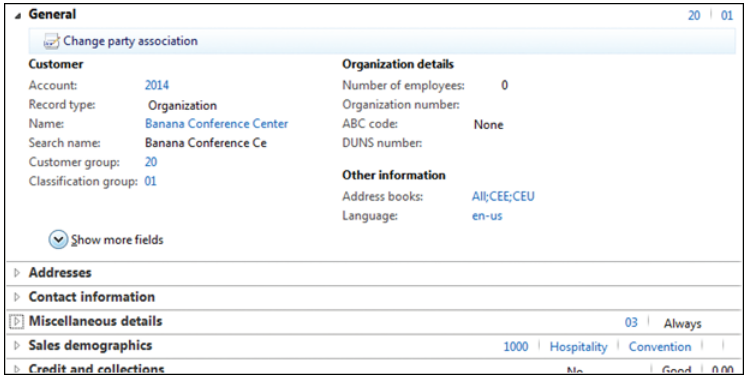


FIGURE 6-10 FastTabs.

To ensure that FastTabs are helpful to users, keep them short so that users can see only the information that's necessary, provide descriptive labels, and display only the most important summary fields when the tab is collapsed. For more tips on creating effective FastTabs, see Chapter 5.

Grid

Use a *Grid* control to display a collection of records that are associated with the primary record on the form. For example, you could use a *Grid* control to display contacts or addresses for a customer.

If you do not want the *Grid* control to take up the entire form, you can control the size by using the *VisibleRows* property. For example, when displaying the addresses of a customer, many customers will only have one or two addresses. Setting *VisibleRows* to *3* is one way to display the relevant information while taking up minimal space.

Input controls

You can use input controls in either a bound or an unbound manner. Input controls can be bound to either fields or methods.

Field-bound controls

Input controls represent the fields on a form. To create input controls that are bound to fields, you can manually add controls to the form and then bind them to fields in the data source. Another alternative is to drag fields from the data source and drop them onto the form, as in the following procedure:

1. Right-click the *Form.Data Sources* node, and then click Open In New Window.
2. Place the new window containing the form data sources next to the original AOT window.

3. Drag the fields you want from the data sources and drop them into the appropriate location in a Group, TabPage, or Grid control on the form. This action creates the appropriate type of input control (*StringEdit* for strings, *IntEdit* for integers, and so on), and binds it to the data source field.

Method-bound controls

If you bind an input control to a display method, you can present data that is processed or created through code. You should place display methods that relate to a particular table on that table instead of on a form data source.

When binding a control to a display method, use the *Datasource* and *Datamethod* properties to point at the appropriate display method.

Display methods use the *display* keyword in the method declaration. The best examples of display methods are those that already exist in Microsoft Dynamics AX. Use the search capability in the AOT to find example display methods on tables by looking for the *display* keyword followed by a space (*display*).

The standard display method format is as follows:

```
display SomeEDT myDisplayMethod()
{
    //Code here...
    return "returnValue";
}
```

Unbound controls

You can add input controls such as *StringEdit* and *IntEdit* to a form and manipulate them through X++ code to provide the user experience you want. An example of this is seen in the AxdWizard form. Unbound controls can also be used to provide a custom filtering experience. An example of this is seen in the SalesLineBackOrder form.

ManagedHost control

If the predefined controls provided with Microsoft Dynamics AX do not meet a specific need, or when a using prebuilt component would save time, you can use an externally-created control. With the *ManagedHost* control, you can use .NET controls on Microsoft Dynamics AX forms. The *ManagedHost* control is the preferred solution when you need an externally-created control because of the ease of use it provides.

To use a .NET control within a Microsoft Dynamics AX form, the AOT must contain a reference to the .NET assembly that contains the control. To add new .NET controls, add the assembly or assemblies that contain the controls to the *Reference* node of the AOT by right-clicking that node and then clicking Add Reference.

To reference that .NET control within a Microsoft Dynamics AX form, add a *ManagedHost* control and then use the Managed Control Selector dialog box to select the control you want. Right-click the new control to subscribe to events.

Try this simple example to add a .NET button to a form:

1. In the AOT, right-click the *Forms* node, and then click New.
2. Right-click the *Design* node, and then point to New Control > ManagedHost.
3. In the Managed Control Selector dialog box, in the top grid, select the *System.Windows.Forms* assembly, and then in the Controls grid, select Button, and then click OK.

Note The *System.Windows.Forms* assembly is referenced by default, so you do not need to add a reference.

4. Set the name of the control to ManagedButton.
5. Right-click the ManagedButton control to open the Events dialog box, and then add the *Click* event.
6. Expand the *Methods* node for the form, open the *init* method, and replace the existing code with the following to set the text for the button:

```
public void init()
{
    super();
    _ManagedButton_Control = ManagedButton.control();
    _ManagedButton_Control.add_Click(new ManagedEventHandler(this, 'ManagedButton_
Click'));
    _ManagedButton_Control.set_Text("Managed button");
}
```

7. Open the code for the *Click* method, and then replace the existing code with the following to display text in the InfoLog:

```
void ManagedButton_Click(System.Object sender, System.EventArgs e)
{
    info("Managed button clicked");
}
```

Run the form, and then click the button. The result is shown in Figure 6-11.

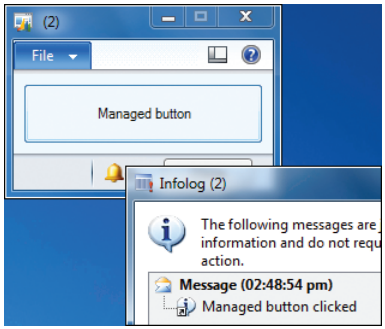


FIGURE 6-11 *ManagedHost* control example.

Other controls

You can add additional types of controls to a form to provide additional information or interactivity. For example, the static text control can provide instructional text to guide a user through a process, and the Window control can display images that help inform the user about a product or service. For more information, see "Controls in Microsoft Dynamics AX" at *http://msdn.microsoft.com/en-us/library/gg881259*.

Using parts

You use a part to retrieve and show data that is related to the selected record on the host form. Parts can be used in the FactBox pane of any form or in the preview pane of a list page.

Types of parts

The following types of parts are available:

- Info parts are displayed like forms at runtime. At design time, info parts use a simplified set of metadata that allows them to be displayed in both the Microsoft Dynamics AX client and Enterprise Portal web client. Info parts have simple styling and are essentially a collection of data fields from the specified query. Info parts can define a set of actions to display below the data fields. Preview panes for list pages are always modeled as info parts.
- Cue groups are a collection of cues. Cues are a mechanism for showing the count of the records from a query. Often, the query used for the cue is restricted based on the record currently shown on the host form. A cue contains three things: a query that provides the count, a *MenuItemName* property that specifies the action to take when a user clicks a cue, and a label that informs the user what the count is for (if none is provided, the MenuItem label is used). For more information about how to use cues, see Chapter 5.

- Form parts are pointers to existing forms that can be displayed as FactBoxes. The *Form* property specifies the form to display, and the *Caption* property provides the title caption for the FactBox. When you are building a form to display as a form part, set the *Style* property to *FormPart*, the *ViewEditMode* property to *View*, and the *Width* property to *ColumnWidth* to ensure correct styling.

Reference a part from a form

The *Parts* node for each form references the parts that are used to display data related to the record being displayed by the form. Within the *Parts* node, you create part references that ensure the correct context.

To create a standard part reference:

1. Set the *MenuItemName* property to the *MenuItem* that specifies the part. A *MenuItem* is used to reference each part to ensure that standard *MenuItem*-based security can be applied.
2. Set the *DataSourceName* and *DataSourceRelationName* properties to specify the correct data relation (dynalink) to use between the host form and the part.
3. Set the *PartLocation* property to indicate whether the part should be displayed as a FactBox (the default) or as a preview pane.
4. (Optional. For list pages only.) Set the *DisplayTarget* property to indicate whether the part will be displayed in the Microsoft Dynamics AX client, Enterprise Portal, or both.
5. (Optional.) Set the *Visible* property to hide the FactBox by default. The FactBox is still available for users if they choose to show it.

Adding navigation items

To give users access to the forms you create, you add references to them on menus.

MenuItem

In Microsoft Dynamics AX, a *MenuItem* is a modeled pointer to another resource such as a form, class, or report. You define *MenuItem* metadata in the *Menu Items* node of the AOT. The *Menu Items* node has three subnodes that are used for categorization purposes. The *Display*, *Action*, and *Output* types usually reference forms, classes, and reports, respectively. However, a common exception is to have a display *MenuItem* reference a class that is used to initialize and open a form.

Menu

In Microsoft Dynamics AX, a *Menu* is a structured collection of references to *MenuItems* and other *Menus*. The navigation pane, area pages, and address bar are mechanisms for exposing the menu metadata that you define in the *Menus* and *Menu Items* nodes of the AOT. The module menus are defined in the *Menus**MainMenu* node of the AOT. You can follow the menu structure from that starting point. For example, the *Accounts Receivable* module is represented by the *Menus**MainMenu**AccountsReceivable MenuReference* and is defined in *Menus**AccountsReceivable*.

When adding a *Menu* item to a menu, ensure that the *IsDisplayedInContentArea* property is set appropriately. For list pages and content pages that are displayed in the client, set this property to *Yes* so that the address bar is populated correctly.

Menu definitions

In previous releases of Microsoft Dynamics AX, forms were generally specific to a single module. However, in Microsoft Dynamics AX 2012, the application has been reorganized to be more role-specific. As a result, several new modules were created, such as Sales and Marketing and Inventory and Warehouse Management. Many commonly-used forms are now found in multiple modules; for example, the Customers and Sales Orders forms are now located in both the Accounts Receivable and Sales and Marketing modules.

When defining a module menu or adding items to an existing module menu, try to follow the standard groupings that are used in other menus:

- ***Common*** Contains the most commonly-accessed forms in the module. The *Common* group usually contains links to list pages.
- ***Periodic*** Contains links to secondary data forms.
- ***Inquiries*** Contains links to forms that provide read-only views of data that is related to the current module.
- ***Reports*** Contains links to reports.
- **Setup** Contains links to setup forms, including the parameters forms. Sometimes this group also contains secondary data forms.

The primary list pages listed in the *Common* group should be accompanied by secondary list pages. A secondary list page is a list page that adds ranges (filters) to a primary list page. You can implement a secondary list page as a menu item that points at the primary list page form but also specifies a query that adds a filter.

Customizing forms with code

You should customize forms with code only when the result cannot be accomplished by customizing metadata. When you customize forms by using metadata, upgrades are easier. Metadata change conflicts are easier to resolve, whereas code change conflicts need deeper investigation that sometimes involves creating a new merged method that attempts to replicate the behavior of the two original methods.

When you customize Microsoft Dynamics AX, the following ideas might provide good starting points for investigation:

- Use examples in the base Microsoft Dynamics AX 2012 application by using the *Find* command on the *Forms* node in the AOT (Ctrl+F).
- Refer to the system documentation entries (*AOT\System Documentation*) for information about system classes, tables, functions, enumerations, and other system elements that are implemented in the AX kernel.
- Add a debug breakpoint in the *init* method for the form when you are looking for a suitable location for your customization code. Step through the execution of the method overrides. Note that control events (such as *Clicked*) do not trigger debugging breakpoints. You must explicitly add the *breakpoint* keyword to the X++ code for the debugger to stop in these methods.

For simpler code maintenance, follow these guidelines:

- Use the field and table functions of *fieldNum,* such as *fieldNum(SalesTable, SalesId)*, and *tableNum,* such as *tableNum(SalesTable)*, when working with form data sources.
- Avoid hard-coding strings. Instead, use labels, such as *throw error("@SYS88659"),* and functions such as *fieldStr and tableStr*, which return the names of specified fields or a specified table, respectively.
- Use as few method overrides as possible. Each additional method override has a chance of causing merge issues during future upgrades, patch applications, or code integrations.

Method overrides

By overriding form methods, you can influence the form lifecycle and control how the form responds to some user-initiated events. Table 6-5 describes the most important form methods to override. The most-commonly overridden form methods are *init* and *run*.

TABLE 6-5 Form methods to override.

Method	Description
init	Called when the form is initialized. Prior to the call to *super*, much of the form (*FormRun*) is not initialized, including the controls and the query. This method is commonly overridden to access the form at the earliest stage possible.
run	Called when the form is initialized. Prior to the call to *super*, the form is initialized but isn't visible to the user. This method is commonly overridden to make changes to form controls, layout, and cursor focus.
createRecord	Called when a record is being created in the form. This method is commonly used to intercept the event of any record being created on the form. It is also used to provide specific types for creating inheritance records. For more information, see "Table inheritance" earlier in this chapter.
close	Called when the form is being closed. This method is commonly overridden to release resources and save user settings and selections.
task	Called when the user performs a task or issues a command on the form. The task method contains many of the common tasks for a form.
activate	Called when the form is activated. This method is commonly used to set the company when the form is activated.
closeOk	Called when the user closes the form by using the OK command, such as when the user clicks a *CommandButton* with a *Command* property of *Ok*. This method is commonly overridden in dialog boxes to perform the action the user has initiated.
closeCancel	Called when the user closes the form by using the Cancel command, such as when the user clicks a *CommandButton* with a *Command* property of *Cancel*. This method is commonly overridden in dialog boxes to clean up after the user indicates that an action should be cancelled.
canClose	Called when the form is being closed. This method is commonly overridden to ensure that data is in a valid state before the form is closed. A return value of *false* stops the action and keeps the form open.

By overriding methods on form data sources and form data source fields, you can influence how the form reads and writes data and responds to user-initiated data-related events. Table 6-6 describes the most important form data source methods to override. The most-commonly overridden form data source methods are *init*, *active*, *executeQuery*, *write*, and *linkActive*.

TABLE 6-6 Form data source methods to override.

Method	Explanation
active	Called when the active record changes, such as when the user clicks a different record. This method is commonly overridden to enable or disable buttons based on whether they are applicable to the current record.
create	Called when a record is being created, such as when the user presses Ctrl+N. This method is commonly overridden to change the user interface in response to the creation of a record.
delete	Called when a record is being deleted, such as when the user presses Alt+F9. This method is commonly overridden to change the user interface in response to the deletion of a record.
deleting	Called before a record is deleted. This method is valid only when the *ChangeGroupMode* property on the data source is set to *ImplicitInnerOuter*. This method is commonly overridden to change the user interface in response to the deletion of a record.
deleted	Called after a record has been deleted. This method is valid only if the *ChangeGroupMode* property is set to *ImplicitInnerOuter*. This method is commonly overridden to change the user interface in response to the deletion of a record.

Method	Explanation
executeQuery	Called when the query for the data source executes, such as when the form runs (from the super of the form's *run* method) or when the user refreshes the form by pressing F5 (F5 calls *research*, which calls *executeQuery*). This method is commonly overridden to implement the behavior of a custom filter added to the form.
init	Called when the data source is initialized during the call to the *super* of the form's *init* method. This method is commonly overridden to add or remove query ranges or change dynalinks.
initValue	Called when a record is being created. Record values set in this method count as original values rather than changes. This method is commonly overridden to set the default values of a new record.
leaveRecord	Called when the user moves the focus from one data source join hierarchy to another, which can happen when the user moves between controls. This method is sometimes overridden to respond to the data save operation that will occur, but is recommended that you use the *validateWrite* and *write* methods whenever possible. The *validateWrite* and write methods are called in the call to the super of the *leaveRecord* method.
linkActive	Called when the active method in a dynalinked parent data source is called. This method is commonly overridden to change the user interface to correspond to a different parent record (*element.args.record*).
markChanged	Called when the marked set of records changes, such as when the user multiselects a set of records. This method is commonly overridden to enable or disable buttons that work on a *multiselected* (marked) set of records.
validateDelete	Called when the record is being deleted. This method is commonly overridden to provide form-specific validation of the deletion event. If the method returns a value of *false*, the deletion is stopped. Use the *validateDelete* table method to provide record deletion validation across all forms.
validateWrite	Called when the record is being saved, such as when the user presses the Close or Save buttons or clicks a field that is associated with another data source. This method is commonly overridden to provide form-specific write/save event validation. Returns *false* to stop the *write*. Use the *ValidateWrite* table method to provide record write/save validation across all forms.
write	Called when the record is being saved after validation has succeeded. This method is commonly overridden to perform additional form-specific logic for the write or save events, such as updating the user interface. Use the *write* table method to respond to the *write* and save events for the record across all forms.
writing	Called before a record is written to the database. This method is valid only if the *ChangeGroupMode* property is set to *ImplicitInnerOuter*. Use the *writing* table method to respond before the *write* and *save* events for the record across all forms.
written	Called after a record has been written to the database. This method is valid only if the *ChangeGroupMode* property is set to *ImplicitInnerOuter*. Use the *written* table method to respond after the *write* and *save* events for the record across all forms.

Table 6-7 describes the methods to override for fields in form data sources. The most-commonly overridden method for form data source fields is the *modified* method.

TABLE 6-7 Field methods to override.

Method	Explanation
modified	Called when the value of a field changes. This method is commonly overridden to make a corresponding change to the user interface or to change other field values.
lookup	Called when the user clicks the Lookup button for the field. This method is commonly overridden to build a custom lookup form. Use this method sparingly to provide lookup behavior form a specific form. Instead, consider using the *EDT.FormHelp* property to provide lookup capabilities that can be shared across multiple forms.

Method	Explanation
validate	Called when the value of a field changes. This method is commonly overridden to perform form-specific validation needed prior to changing the value of a field. Use a return value of *false* to stop the change. Use the *validateField* table method to provide field validation across all forms.

Auto variables

When X++ code executes in the scope of a form, form-specific *Auto* variables are created to help developers access important objects related to the form. These variables are read-only and are described in Table 6-8.

TABLE 6-8 Form-specific *Auto* variables.

Variable	Description

element	Variable that provides easy access to the *FormRun* object that is in scope. This variable is commonly used to call methods or change the design.
	Example:
	`element.args().record().TableId == tablenum(SalesTable)`
	`name = element.design().addControl(FormControlType::String, "X");`

DataSourceName (for example, *SalesTable*)	Variable that provides easy access to the current active record and table in each data source. This variable is commonly used to call methods or *get* or *set* properties for the current record. Example: `if (SalesTable.type().canHaveCreditCard())`
DataSourceName_DS (for example, *SalesTable_DS*)	Variable that provides easy access to each data source. This variable is commonly used to call methods or *get* or *set* properties for the data source. Example: `SalesTable_DS.research();`

DataSourceName_Q (for example, *SalesTable_Q*)	Variable that provides easy access to each data source's *Query* object. This variable is commonly used to access the data source query to add ranges before the query executes. This variable is equivalent to *SalesTable_DS.query()*.
	Example:
	`rangeSalesLineProjId = salesLine1_q.dataSourceTable(tablenum(SalesLine)).addRange(fieldnum(SalesLine, ProjId));`
	`rangeSalesLineProjId.value(ProjTable.ProjId);`

DataSourceName_QR (for example, *SalesTable_QR*)	Variable that provides easy access to each data source's *QueryRun* object, which contains a copy of the query that was most recently executed. The query inside the *QueryRun* object is created during the call to the *FormDataSource ExecuteQuery* method. This variable is commonly used to access the query that was executed so that query ranges can be inspected. This variable is equivalent to *SalesTable_DS.queryRun()*. Example: `SalesTableQueryBuildDataSource = SalesTable_QR.query().dataSourceTable(tablenum(SalesTable));`
ControlName (for example, *SalesTable_SalesId*)	Variable created for each control whose *AutoDeclaration* is set to *Yes*. This variable is commonly used to access controls that are not bound to a data source field, such as the fields used to implement custom filters. Example: `backorderDate.dateValue(systemdateget());`

Business logic

After the form structure is complete, add calls to business logic by using *MenuItem* references or by using explicit code in method overrides or button clicks. Try to keep explicit code on the form to a minimum because any code that is written on the form cannot be used in other forms, reports, services, or form classes. If you need to add business logic, place it in separate classes when possible to allow it to be used with multiple forms.

To reference business logic in classes:

1. Put a static *main* method on the class.
2. Add the code to the *main* method that starts the business logic.
3. Create an *Action MenuItem* that references the class.
4. Add a *MenuItemButton* on the form that points at the *Action MenuItem*.

For an example, you can follow these steps, using the following code inside the main method:

```
static void main(Args args)
{
    print "Hello World";
    pause;
}
```

Once control has been passed to the class, the *args* method can provide contextual information that may be useful when the class is called from multiple forms.

Custom lookups

Lookups for table references are provided automatically by the client framework and are sufficient for the large majority of scenarios. Automatic lookups are generated by using metadata from the target table. To get the fields to use for the lookup form, the framework first checks the *AutoLookup* field group on the table. If that field group is empty, the framework checks the *AutoIdentification* field group. If that field group is empty, the *TitleField1* and *TitleField2* fields from the table are used for the lookup.

Automatic lookups generated by the framework perform in the ideal way for usability. If you choose to create your own custom lookup form for a given table, you should use the same pattern so that the behavior is consistent.

Creating a simple custom lookup is simple, especially if you want it to be used for all lookups for the target table type. Model a simple form with a *Grid* control to display the records, and then use the form name as the value for the *FormHelp* property on the EDT for the foreign key field. This works for both regular foreign keys and surrogate foreign keys. When the *FormHelp* property is set, then the custom lookup form will be used instead of an automatically-generated lookup.

In some scenarios, such as query modification or custom record selection, you might want to provide logic that runs before or after the lookup form is loaded. In these cases, you can use the *FormAutoLookupFactory* class. This class is implemented in the kernel and exposes much of the same functionality, such as initial positioning and filtering, to allow custom lookups to behave consistently. For an example in the application, examine the *HCMWorkerLookup* form and class. Looking at the class, you will notice that there are many different scenarios in which this form can be loaded. The different methods on the *FormAutoLookupFactory* class are called in each case. There is also corresponding code on the form that handles these cases, such as the code to set the *SelectMode* for the different types of source control.

Integrating with the Microsoft Office client

With the Microsoft Dynamics AX 2012 Office Add-ins, users can pull Microsoft Dynamics AX data into Microsoft Excel for ad hoc and predefined reporting, push data from Excel into Microsoft Dynamics AX for data entry, and generate Microsoft Word documents for sharing data with others.

This section describes how to make data sources available to the Office Add-ins, and then provides an overview of how to create Excel and Word templates and make them available to users.

Make data sources available to Office Add-ins

Before the Office Add-ins can consume data from Microsoft Dynamics AX, you must make the appropriate services and queries available as data sources.

Make a service available

To make a service available:

1. In the AOT, under the *Services* node, right-click the service that you want to make available, and then click Add-ins > Register Service.
2. In the client, click System Administration > Setup > Services and Application Integration Framework > Inbound Ports, and then do the following:
   - Create a new inbound port.
   - Select the service operations to add to the port.
   - Activate the port.
3. In the client, navigate to Organization Administration > Setup > Document Management > Document Data Sources.
4. In the Document Data Sources form, do the following:
   - Create a new document data source.
   - Select the module that is associated with the data source.

- Set the *Type* field to *Service*.
- Select the inbound port that you just added as the data source.
- Activate the new document data source.

Make a query available

To make a query available:

1. Define a new query in the AOT, if necessary.
2. In the client, click Organization Administration > Setup > Document Management > Document Data Sources.
3. In the Document Data Sources form, do the following:
   - Create a new document data source.
   - Select the module that the data source (the query) is associated with.
   - Set the *Type* field to *Query*.
   - Select the query that you want to use as the data source.
   - Activate the new document data source.

Build an Excel template

After you make the appropriate queries and services available as document data sources to the Office Add-ins, you can create Excel templates that access data through them. Users can then use these to view and analyze Microsoft Dynamics AX data in Excel, using Excel features such as conditional formatting, PivotTables, and calculated fields. If the workbook uses service data sources, users can modify the data in the workbook and then publish those data changes back to Microsoft Dynamics AX.

A template can be as simple as a listing of the latest sales orders or as complex as an executive digital dashboard. Figure 6-12 shows an example of an Excel template.

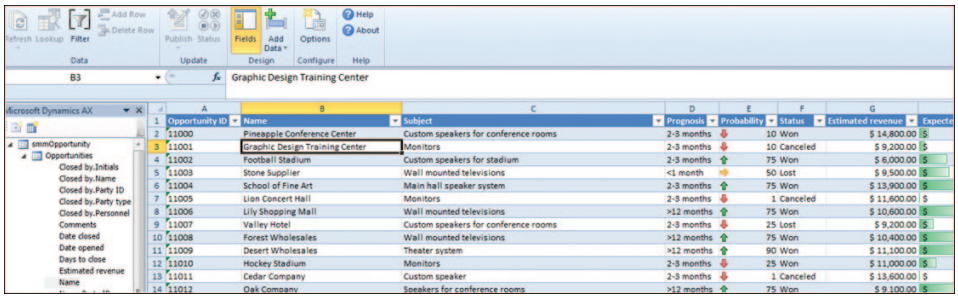


FIGURE 6-12 Excel template with Microsoft Dynamics AX data.

From within an Excel workbook, do the following to access data from Microsoft Dynamics AX:

1. Open the Options dialog box from the Microsoft Dynamics AX tab of the Ribbon to ensure that the appropriate server and port connection information is present.
2. Click Add Data, and then select the appropriate query and service data sources.
3. Double-click or drag and drop fields from the field chooser to add them to the worksheet.
4. Refresh the worksheet to verify the data that is being retrieved from Microsoft Dynamics AX and added to the workbook. If the dataset is too large, use the Filter option in the Ribbon to add a filter.

Before providing the workbook to other users, do the following:

1. If necessary, add additional filters to restrict the dataset that is returned.
2. Open the Connection Options dialog box and remove the existing connection information, so that the user's connection information is supplied automatically by the Client SDK (using the information contained in the Microsoft Dynamics AX Client Configuration Utility).
3. Save the workbook without connection information.

Build a Word template

You can create Word templates that allow users to generate Word documents that contain Microsoft Dynamics AX data. Figure 6-13 shows an example of a Word template.

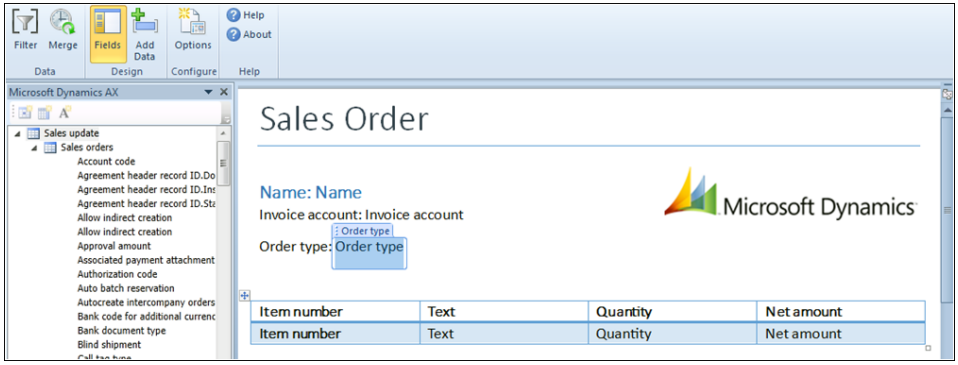


FIGURE 6-13 Word template with Dynamics AX data.

From within a Word document, do the following to access data from Microsoft Dynamics AX:

1. Open the Options dialog box from the Microsoft Dynamics AX tab of the Ribbon to ensure that the appropriate server and port connection information is present.
2. Click Add Data, and then select the appropriate query and service data sources.

3. Double-click or drag and drop fields from the field chooser to add them to the document. If you want to show calculated fields (display methods) on a data source, then right-click that data source in the field chooser and select Show Calculated Fields.
4. Individual field bindings can be added throughout the document. These fields can be interspersed with static text, formatting, images, and other content.
5. Repeated values, like the lines of a Sales Order, can be displayed by inserting a table and then adding field bindings into the first row of that table.
6. Add a filter to select a particular record. When using a template in the "Generate from template" functionality, this record-specific filter is not present.
7. Save the document.
8. Click the Merge button to generate a document from the template.

Before sharing a document template with other users:

1. Add any additional Filters to restrict the dataset returned as needed.
2. Open the connection options dialog and remove the existing connection information so the user's connection information is supplied automatically by the Client SDK (using the information contained in the Client Configuration Utility).
3. Save the document without connection information.
4. Provide your users with a copy of the document.

Add templates for users

Several forms in Microsoft Dynamics AX have a Generate from Template button in the Attachments group of the Action pane. An example of the Customers list page is shown in Figure 6-14.

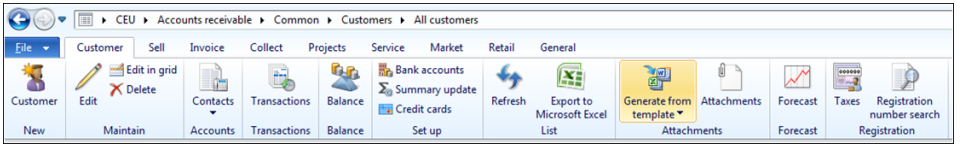


FIGURE 6-14 Generate from Template button on an Action pane.

To add a group of templates as an option for the Generate from Template:

1. Create Word document or Excel workbook templates with a filter that does not restrict the results to a single record.
2. In the client, click Organization Administration > Setup > Document Management > Document Types.

3. Create a new document type, setting the *Class* field to *Template Library*.

4. Set the *Document Library* field to point to the SharePoint folder where the templates are located. Ensure that the URL points to the folder and not a page; for example, *http://myserver/DocumentTemplates/*.

5. Click Synchronize to import the template list and activate the templates.

6. Verify that the templates appear in the Generate from Template list on the form. If the templates are not shown, ensure that the primary data source for the templates matches the primary data source for the form.

If the Generate from Template button is not available on a form, users can still generate a document from a template by opening the Document Handling form (File > Command > Document Handling) and creating a new attachment from the Template Library type.

To add the Generate from Template button to additional forms or list pages:

1. Find the Generate from Template button on the Customers list page and copy it to the form you want to add it to. The path to the button is as follows:

   AOT\Forms\CustTableListPage.Designs\Design\ActionPane:ActionPane\ ActionPaneTab:HomeTab\ButtonGroup:AttachmentsGroup\MenuButton:mbTemplatesButton

2. Edit the *MouseDown* method on the button to pass the correct *TableId* to the *createTemplateOnMenuButton* method by changing the *CustTable.TableId* parameter to point to the correct table; for example, *MyTable.TableId*.

CHAPTER 7

Enterprise Portal

In this chapter

Introduction 195
Enterprise Portal architecture 196
Enterprise Portal components 198
Developing for Enterprise Portal 216
Security 232
SharePoint integration 235

Introduction

With the Microsoft Dynamics AX Enterprise Portal web client, organizations can extend and expand the use of enterprise resource planning (ERP) software to reach out to customers, vendors, business partners, and employees by allowing them to access business applications and collaborate from anywhere.

Users access Enterprise Portal remotely through a web browser or from within a corporate intranet, depending on how Enterprise Portal is configured and deployed. Enterprise Portal serves as the central place for users to access any data, structured or unstructured, such as transactional data, reports, charts, key performance indicators (KPIs), documents, and alerts. For information about the Enterprise Portal user interface, see Chapter 5, "Designing the user experience."

Enterprise Portal also serves as a web platform. It contains a set of default webpages and user roles that you can use as is or modify to meet unique business needs. You can also web-enable, customize, or create new business applications in Microsoft Dynamics AX.

Enterprise Portal in Microsoft Dynamics AX 2012 adds a number of new features to speed up the development of business applications. By using the new model-driven development approach, you can build list pages that work both in Enterprise Portal and the Microsoft Dynamics AX Windows client, reducing development time. New project and control templates in Microsoft Visual Studio enable rapid application development. New metadata settings let you instantly enable common patterns that you previously had to write code to support. For more information, see "What's New: Enterprise Portal for Developers in Microsoft Dynamics AX 2012," at *http://msdn.microsoft.com/en-us/library/gg845087.aspx*.

Enterprise Portal architecture

Enterprise Portal brings the best of Microsoft Dynamics AX, ASP.NET, and Microsoft SharePoint technologies together to provide a rich web-based business application. It combines the rich functionality of SharePoint with the structured business data in Microsoft Dynamics AX.

You can use MorphX to take advantage of the rich programming model to define data access and business logic in Microsoft Dynamics AX. You can build web user controls and define the web user interface elements using Visual Studio. The web controls can contain Microsoft Dynamics AX components like *AxGridView*, as well as standard ASP.NET controls like *TextBox*. The data access and business logic defined in Microsoft Dynamics AX is exposed to the web user controls through data binding, data and metadata application programming interfaces (APIs), and proxy classes.

Figure 7-1 shows the architecture of Enterprise Portal.

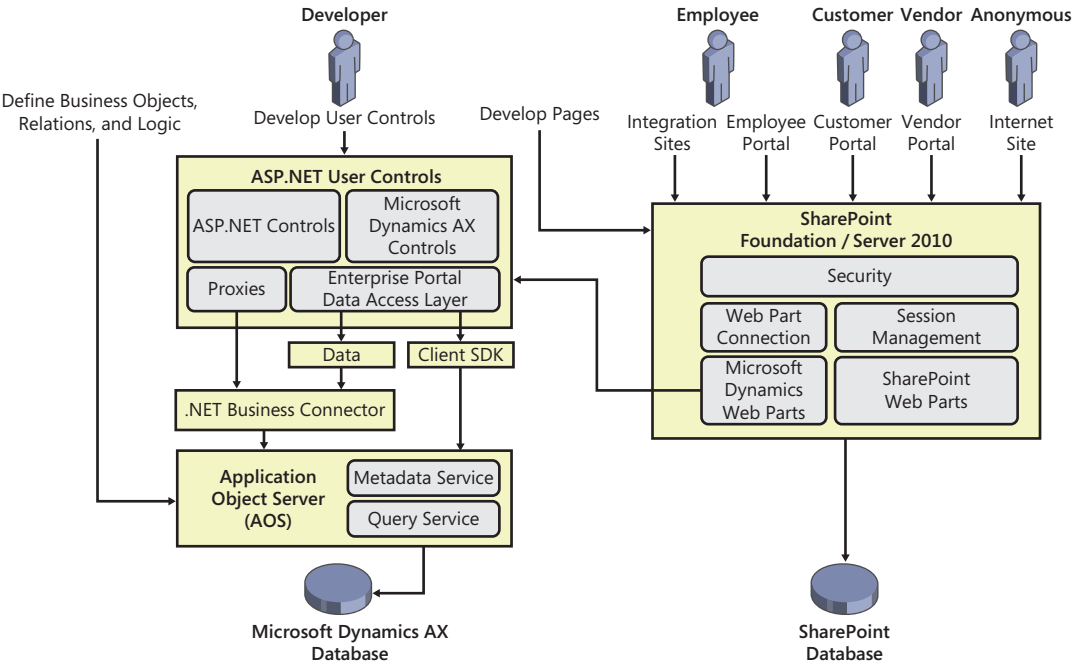


FIGURE 7-1 Enterprise Portal architecture.

Enterprise Portal uses the web part page framework from SharePoint. Web parts are reusable SharePoint components that generate HTML and provide the foundation for the modular presentation of data. By using this framework, you can build webpages that allow easy customization and personalization. The web part page framework also makes it easy to integrate content, collaborate, and use third-party applications. Webpages can contain both Microsoft Dynamics AX web parts and SharePoint web parts.

The Microsoft Dynamics AX web parts present information and expose functionality from Microsoft Dynamics AX. The User control web part can host any ASP.NET web user control,

including the Enterprise Portal web user controls. It can connect to Microsoft Dynamics AX through the Enterprise Portal framework. You can use SharePoint web parts to fulfill other content and collaboration needs. For example, you might have a custom web part that goes out to a site, fetches the latest news about your organization, and displays it. Or you might have a web part that displays data from another SharePoint site within your organization.

The first step in developing or customizing an application on Enterprise Portal is to understand the interactions between the user's browser on the client and Enterprise Portal on the server when the user accesses Enterprise Portal.

The following sequence of interactions occurs when a user accesses an Enterprise Portal page:

1. The user opens the browser on his or her computer and navigates to Enterprise Portal.
2. The browser establishes a connection with the Internet Information Services (IIS) web server.
3. IIS authenticates the user based on the authentication mode being used.
4. After the user is authenticated, SharePoint verifies that the user has permission to access the site.
5. If the user is authorized to access the site, the request is passed to the SharePoint module.
6. SharePoint gets the data about the page from the SharePoint database or the file system. This data consists of information such as the page layout, the master page, the web parts that go on the page, and their properties.
7. SharePoint processes the page by creating and initializing the web parts and applying any properties and personalization data. To display the top navigation bar, the quick launch, and the Action pane, a custom navigation provider gets information from Microsoft Dynamics AX (modules, menus, submenus, and menu items).
8. Enterprise Portal initializes the Microsoft Dynamics AX web parts and starts a web session with the Enterprise Portal framework through the .NET Business Connector to the Application Object Server (AOS).
9. The web framework checks for Microsoft Dynamics AX authorization and then calls the appropriate web handlers in the web framework to process the Enterprise Portal objects that the web parts point to.
10. The User control web part runs the web user control that it references. The web user control connects to Microsoft Dynamics AX through .NET Business Connector and renders the HTML to the web part.
11. The webpage assembles the HTML returned by all of the web parts and renders the page in the user's browser.

As you can see in this sequence, the AOS processes the business logic and data retrieval, ASP.NET processes the user interface elements, and SharePoint handles the overall page layout and personalization. Figure 7-2 shows a graphical representation of this sequence of events.

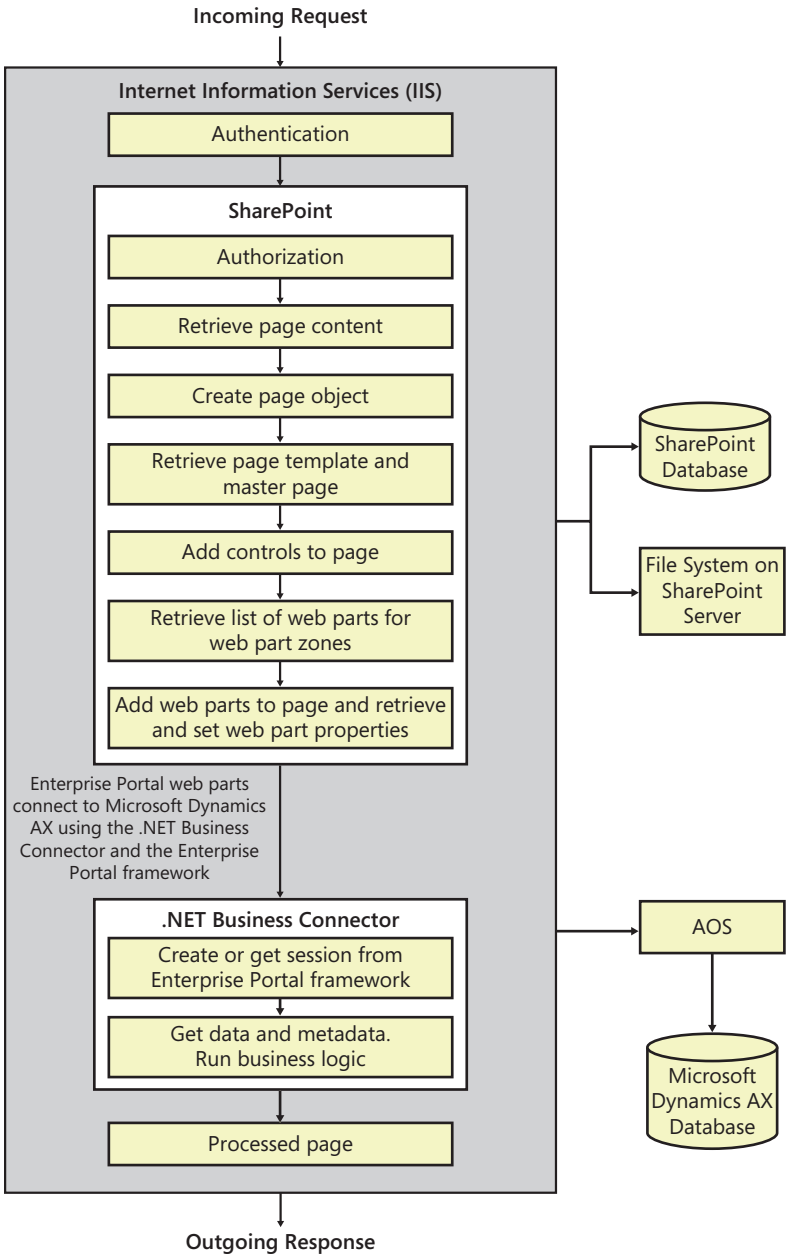


FIGURE 7-2 Enterprise Portal page processing.

Enterprise Portal components

This section describes the components that make up an Enterprise Portal page: web parts, Application Object Tree (AOT) elements, datasets, and Enterprise Portal framework controls.

Web parts

Web parts support customization and personalization and can be integrated easily into a webpage. Enterprise Portal includes a standard set of web parts, shown in Figure 7-3, that expose the business data in Microsoft Dynamics AX.

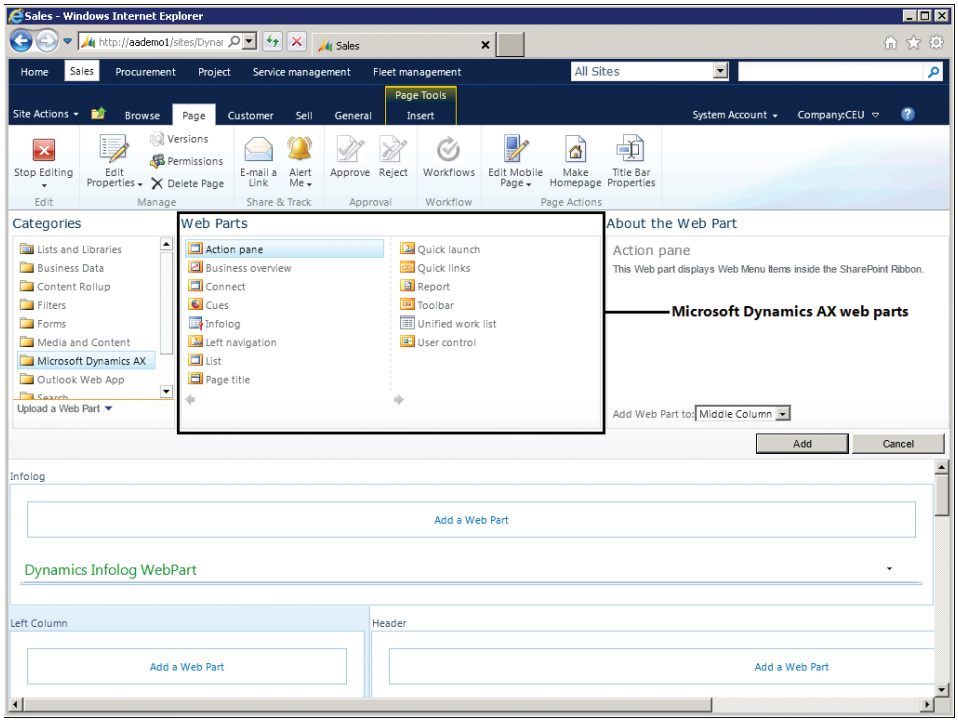


FIGURE 7-3 Adding Microsoft Dynamics AX web parts to a page.

The following Microsoft Dynamics AX web parts are included with Enterprise Portal:

- **Action pane** Used to display the Action pane, which is similar to the SharePoint ribbon. The Action pane points to a web menu in the AOT and displays buttons in tabs and groups to improve their discoverability. You can use the *AxActionPane* control in a web control as an alternative to using the Action pane web part.
- **Business overview** Used to display business intelligence (BI) information such as KPIs and other analytical data in role centers. For more information, see "Chapter 10, "BI and analytics."
- **Connect** Used to display the links to information from the Microsoft Dynamics AX community. This web part is typically used on role center pages.
- **Cues** Used to display numeric information—such as the number of active opportunities, new leads, and so on—visually as a stack of paper. The Cues web part is generally added to Role Center pages and points to a Cue Group in the AOT. For more information about Cues, see Chapter 5.

- **Infolog** Used to display Microsoft Dynamics AX Infolog messages on the webpage. When you create a new web part page by using Enterprise Portal page templates, the Infolog web part is automatically added to the top of the page in the Infolog web part zone. Any error, warning, or information message that Microsoft Dynamics AX generates is automatically displayed by the Infolog web part. If you need to display some information from your web user control in the Infolog web part, you need to send the message through the C# proxy class for the X++ *Infolog* object.
- **Left navigation** Used to display page-specific navigation instead of module-specific navigation. You can use this web part as an alternative to the Quick launch web part, which displays module-specific navigation. This web part points to a web menu in the AOT.
- **List** Used to display the contents of a model-driven list page. When you deploy a model-driven list page to Enterprise Portal, the page template automatically adds the List web part to the Middle Column zone of the page. This web part points to the display menu item for the model-driven list page form.
- **Page title** Used for displaying the page title. When you create a new web part page, the Page title web part is automatically added to the Title Bar zone. By default, the Page title web part displays the text specified in the *PageTitle* property of the *Page Definition* node in the AOT. If no page definition exists, the page name is displayed. You can override this behavior and get the title from another web part on the page by using a web part connection. For example, if you're developing a list page and you want to display information from a record, such as the customer account and name as the page title, you can connect the User control web part that displays the grid to the Page title web part. When the user selects a different record in the customer list, the page title changes to display the currently selected customer account and name.
- **Quick launch** Used for displaying module-specific navigation links on the left side of the page. When you create a new web part page, the Quick launch web part is automatically added to the Left Column zone if the template that you choose has this zone. The Quick launch web part displays the web menu set in the *QuickLaunch* property of the corresponding web module in the AOT. All pages in a given web module (subsite) display the same navigation options in the left pane.
- **Quick links** Used to display a collection of links to frequently used menu items and external websites. This web part is generally added to Role Center pages.
- **Report** Used to display Microsoft SQL Server Reporting Services (SSRS) reports for Microsoft Dynamics AX.
- **Toolbar** Used to display a toolbar on the page in a location that you select. For example, you can use a Toolbar web part to place Add, Edit, and Remove buttons for a grid control right above that grid control. The Toolbar web part points to a web menu in the AOT. Alternatively, you can use an *AxToolbar* control in a web user control instead of the Toolbar web part.

- **Unified work list** Used to display workflow actions, alert notifications, and activities. It is generally added on Role Center pages. For more information, see Chapter 8, "Workflow in Microsoft Dynamics AX."
- **User control** Used for hosting any ASP.NET control, including the Microsoft Dynamics AX web controls that you develop. This web part points to a managed web content item that identifies the web user control. The User control web part can both pass and consume record context information to and from other web parts. To do that set the role of the User control web part as *Provider*, *Consumer*, or *Both*, and then connect it to other web parts. User control web parts automatically use AJAX, which allows them to update the content that they display without having to refresh the entire page.

AOT elements

The AOT contains several elements that are specific to Enterprise Portal, in addition to other programming elements such as forms, classes, and tables. For more information about the elements that are available for creating Enterprise Portal pages, see Chapter 1, "Architectural overview."

Datasets

You use datasets to define the data access logic. A dataset is a collection of data usually presented in tabular form. Datasets bring the familiar data and programming model from Microsoft Dynamics AX forms together with ASP.NET data binding. In addition, datasets offer an extensive X++ programming model for validating and manipulating the data when create, read, update, or delete operations are performed in Enterprise Portal. You can use the *AxDatasource* control to access datasets to display and manipulate data from any ASP.NET control that supports data binding.

You create datasets by using MorphX. A dataset can contain one or more data sources that are joined together. A data source can point to a table or a view in Microsoft Dynamics AX, or you can join data sources to display data from multiple tables as a single data source. To do this, you use inner or outer joins. To display parent-child data, you use active joins. To display data from joined data sources or from parent-child datasets, you use dynamic dataset views (*DataSetView* class). With a view-based interface, tables are accessed through dynamic dataset views instead of directly. You can access inner-joined or outer-joined tables through only one view, which has the same name as the primary data source. Two views are available with active-joined data sources: one with the same name as the parent data source, and another with the same name as the child data source. The child data source contains records related only to the current active record in the parent data source.

Each dataset view can contain zero or more records, depending on the data. Each dataset view also has a corresponding special view, which contains just the current, single active record. This view has the same name as the original view with the suffix *_Current* appended to the view name. Figure 7-4 shows the dataset views inside a dataset, along with the data binding.

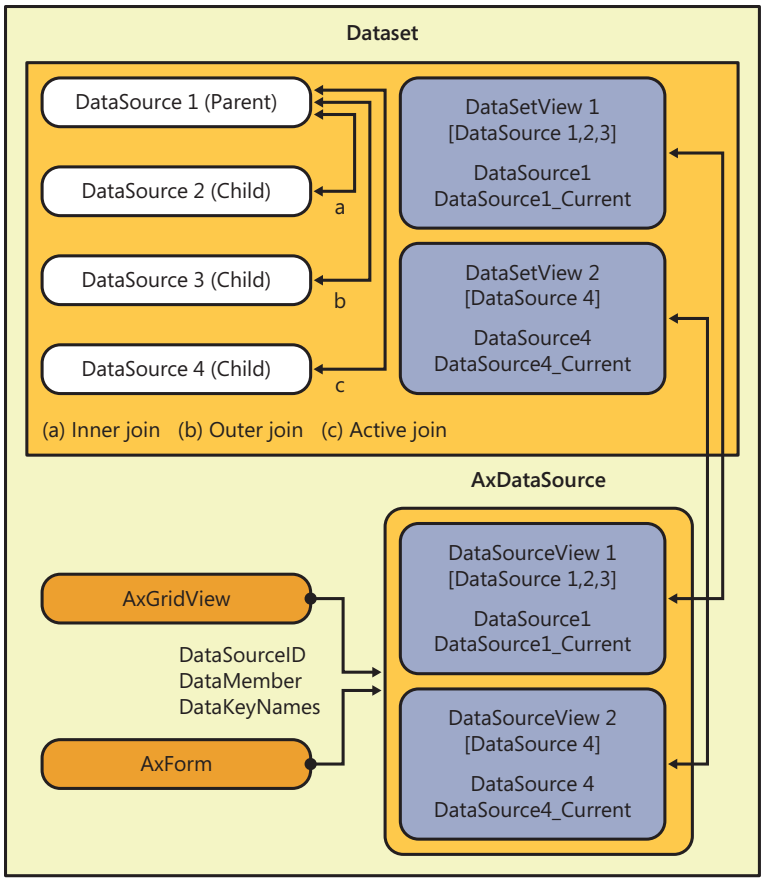


FIGURE 7-4 Enterprise Portal dataset views.

As mentioned earlier, datasets offer an extensive and familiar X++ programming model. Some of the methods used frequently include *init*, *run*, *pack*, and *unpack*:

- ***init*** The *init* method is called when initializing a dataset. This method is called immediately after the *new* operator and creates the run-time image of the dataset. Typical uses of *init* include initializing variables and queries, adding ranges to filter the data, and checking the arguments passed.
- ***run*** The *run* method is called after the dataset is initialized and opened, and immediately after *init*. Typical uses of *run* include conditionally setting the visibility of fields, changing the access level on fields, and modifying queries.
- ***pack*** The *pack* method is called after the dataset is run. You generally implement the *pack-unpack* pattern to save and store the state of an object, which you can later reinstantiate. A typical use of *pack* is to persist a variable used in the dataset between postback actions for user controls.

- ***unpack*** The *unpack* method is called if a dataset was previously packed and is later accessed. If a dataset was previously packed, you do not call *init* and *run*. Instead, you only call *unpack*.

Data sources within a dataset also include a number of methods that you can override. These methods are similar to those in the *FormDataSource* class in the Microsoft Dynamics AX client. You can use them to initialize default values and to validate values and actions. For more information about these events, such as when they are executed and common usage scenarios, see the topic "Methods on a Form Data Source" on MSDN (*http://msdn.microsoft.com/en-us/library/aa893931.aspx*).

Enterprise Portal framework controls

The Enterprise Portal framework has a built-in set of controls that you can use to access, display, and manipulate Microsoft Dynamics AX data.

AxDataSource

The *AxDataSource* control extends *DataSourceControl* in ASP.NET to provide a declarative and data-store-independent way to read and write data from Microsoft Dynamics AX. Datasets that you create in the AOT are exposed to ASP.NET through the *AxDataSource* control. You can associate ASP.NET data-bound user interface controls with the *AxDataSource* control through the *DataSourceID* property. By doing so, you can connect and access data from Microsoft Dynamics AX and bind it to the control without specific domain knowledge of Microsoft Dynamics AX.

The *AxDataSource* control is a container for one or more uniquely named views of type *AxDataSourceView*. The *AxDataSourceView* class extends the Microsoft .NET Framework *DataSourceView* class and implements the functionality to read and write data. A data-bound control can identify the set of capabilities that are enabled by properties of *AxDataSourceView* and use it to show, hide, enable, or disable the user interface components. *AxDataSourceView* maps to the dataset view. The *AxDataSource* control automatically creates *AxDataSourceView* objects based on the dataset that it references. The number of objects created depends on the data sources and the joins that are defined for the dataset. You can use the *DataMember* property of a data-bound control to select a particular view.

The *AxDataSource* control also supports filtering records within and across other *AxDataSource* controls and data source views. When you set the active record on the data source view within an *AxDataSource* control, all child data source views are also filtered based on the active record. You can filter across *AxDataSource* controls by using record context. With record context, one *AxDataSource* control acts as the provider of the context, and one or more *AxDataSource* controls act as consumers. An *AxDataSource* control can act as both a provider and a consumer. When the active record changes on the provider *AxDataSource* control, the record context is passed to other consuming *AxDataSource* controls and they apply that filter also. You use the *Role* and *ProviderView* properties of the *AxDataSource* control to specify whether an *AxDataSource* control is a provider, a consumer, or both. A web user control can contain any number of *AxDataSource* controls; however, only one can be a provider. Any number can be consumers.

You can use the *DataSetViewRow* object to access the rows in a *DataSetView*. The *GetCurrent* method returns the current row, as shown in the following example:

```
DataSetViewRow row = this.AxDataSource1.GetDataSourceView("View1").DataSetView.GetCurrent();
```

The *GetDataSet* method on the *AxDataSource* control specifies the dataset to bind. The *DataSetRun* property provides the run-time instance of the dataset, and you can use the *AxaptaObjectAdapter* property to call methods defined in the dataset.

```
this.AxDataSource1.GetDataSet().DataSetRun.AxaptaObjectAdapter.Call("method1");
```

AxForm

With the *AxForm* control, you can allow users to create, view, and update a single record. This control displays a single record from a data source in a form layout. It is a data-bound control with built-in data modification capabilities. When you use *AxForm* with the declarative *AxDataSource* control, you can easily configure it to display and modify data without having to write any code.

The *DataSourceID*, *DataMember*, and *DataKeyNames* properties define the data-binding capabilities of the *AxForm* control. *AxForm* also provides properties to auto-generate action buttons and to set their text and mode. You set the *UpdateOnPostback* property if you want the record cursor to be updated at postback so that other controls can read the change. *AxForm* also provides *before* and *after* events for all of the actions that can be taken on the form. You can write code in these events to customize the user interface or provide application-specific logic.

AxMultiSection

The *AxMultiSection* control acts as a container for a collection of *AxSection* controls (see the following section). All *AxSection* controls within an *AxMultiSection* control are rendered in a stacked set of rows, which users can expand or collapse. You can configure *AxMultiSection* so that only one section is expanded at a time. In this mode, expanding a section causes it to become active, and any previously expanded section is collapsed. To enable this behavior, set the *ActiveMode* property to *true*. You can then use the *ActiveSectionIndex* property to get or set the active section.

AxSection

AxSection is a generic container for other controls. You can place any control in an *AxSection* control. Each *AxSection* control includes a header that contains the title of the section and a button that allows the user to expand or collapse the section. *AxSection* provides properties to display or hide the header and border. Through events exposed by *AxSection,* you can write code that runs when the section is expanded or collapsed. The *AxSection* control can be placed only within an *AxMultiSection* control.

AxMultiColumn

The *AxMultiColumn* control acts as a container for a collection of *AxColumn* controls. All *AxColumn* controls within an *AxMultiColumn* control are rendered as a series of columns. The *AxMultiColumn* control makes it easy to create a multicolumn layout that optimizes the use of screen space. An *AxMultiColumn* control is usually placed within an *AxSection* control.

AxColumn

AxColumn is a generic container for other controls. You can place any control inside *AxColumn*. However, the *AxColumn* control can be placed only within an *AxMultiColumn* control.

AxGroup

The *AxGroup* control contains the collection of bound fields that displays the information contained in a record. You can place an *AxGroup* control inside an *AxSection* or *AxColumn* control.

Figure 7-5 shows an Enterprise Portal page containing several of the controls that have been discussed so far.

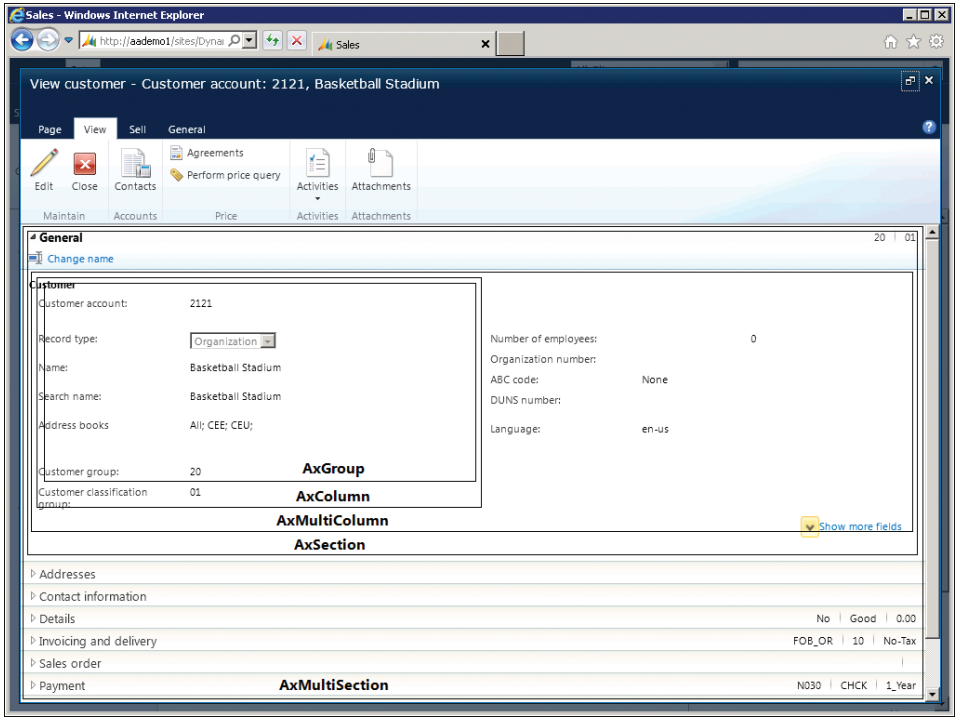


FIGURE 7-5 Enterprise Portal details page with section, column, and group controls.

The following are some high-level control hierarchies for different form layouts. The first one is a commonly used pattern in Enterprise Portal. It displays two expandable sections, one below the other. Each section displays fields in two columns next to each other. If you want to display additional sections or columns, you can add additional *AxSection* and *AxColumn* controls.

```
<AxMultiSection>
    <AxSection>
        <AxMultiColumn>
            <AxColumn>
                <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
            </AxColumn>
            <AxColumn>
                < AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
            </AxColumn>
        </AxMultiColumn>
    </AxSection>
    <AxSection>
        <AxMultiColumn>
            <AxColumn>
                < AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
            </AxColumn>
            <AxColumn>
                < AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
            </AxColumn>
        </AxMultiColumn>
    </AxSection>
</AxMultiSection>
```

The following layout displays two expandable sections, one below the other and in a single column:

```
<AxMultiSection>
    <AxSection>
        <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
        <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
    </AxSection>
    <AxSection>
        <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
        <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
    </AxSection>
</AxMultiSection>
```

The following layout is for an ASP.NET wizard with two steps:

```
<asp:Wizard>
    <WizardSteps>
        <asp:WizardStep>
            <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
        </asp:WizardStep>
        <asp:WizardStep>
            <AxGroup><Fields>BoundFields or TemplateFields…</Fields> </AxGroup>
        </asp:WizardStep>
    </WizardSteps>
</asp:Wizard>
```

AxGridView

The *AxGridView* control displays the values from a data source in a tabular format. Each column represents a field and each row represents a record. The *AxGridView* control extends the ASP.NET *GridView* control to provide selection, grouping, expansion, row filtering, a context menu, and other enhanced capabilities.

AxGridView also includes built-in data modification capabilities. By using *AxGridView* with the declarative *AxDataSource* control, you can easily configure and modify data without writing code. *AxGridView* also has many properties, methods, and events that you can easily customize with application-specific user interface logic.

Table 7-1 lists some of the *AxGridView* properties and events. For a complete list of properties, methods, and events for *AxGridView,* see "AxGridview," at *http://msdn.microsoft.com/en-us/library/cc584514.aspx*.

TABLE 7-1 *AxGridView* properties and events.

Property or event	Description
AllowDelete	If enabled, and if the user has delete permission on the selected record, *AxGridView* displays a Delete button that allows the user to delete the selected row.
AllowEdit	If enabled, and if the user has update permission on the selected record, *AxGridView* displays Save and Cancel buttons on the selected row and allows the user to edit and save or cancel edits. When a row is selected, it automatically goes into edit mode, and edit controls are displayed for all columns for the selected record for which the *AllowEdit* property for the column is set to *true*.
AllowGroupCollapse	If grouping is enabled, this setting allows the user to collapse the grouping.
AllowGrouping	If set to *true* and a group field is specified, the rows are displayed in groups and sorted by group field. Page size is maintained, so one group can span multiple pages.
AllowSelection	If set to *true*, the user can select a row.
ContextMenuName	If *ShowContextMenu* is enabled, *ContextMenuName* specifies the name of the web menu in the AOT to be used as a context menu when the user right-clicks the selected row.
DataBound	An event that is triggered after a control binds to the data source.
DataMember	Identifies the table or dataset that the grid binds to.
DataSourceID	Identifies the *AxDataSource* control that is used to get the data.
DisplayGroupFieldName	If set to *true*, the *GroupFieldName* is displayed in the group header text of the group view.
ExpansionColumnIndexesHidden	A comma-separated list of integers that represents the indexes of the columns to hide from the expansion row. The column indexes start with 1.
ExpansionTooltip	The tooltip displayed on the expansion row link.
GridColumIndexesHidden	A comma-separated list of integers that represents the indexes of the columns to hide from the grid row. The column indexes start with 1.
GroupField	Specifies the data field that is used to group the rows in a group view of the grid control. This property is used only when *AllowGrouping* is set to *true*.
GroupFieldDisplayName	Gets or sets the display name for the group field in the group header text of the group view. If a name isn't specified, by default the label of the *GroupField* is used.

Property or event	Description
*Row**	Events that are triggered in response to the various actions performed on a row. For example, *RowCommand* is triggered when a button in the row is clicked, and *RowDeleted* is triggered when the Delete button in the row is clicked.
SelectedIndexChanged	An event that is triggered when a row is selected in the grid.
ShowContextMenu	If set to *true*, displays the context menu specified by *ContextMenuName* when the user right-clicks the selected row.
ShowExpansion	Specifies whether an expansion is available for each row in the grid. When set to *true*, the expansion row link is displayed for each row in the grid.
ShowFilter	If set to *true*, displays a filter control above the grid.

AxHierarchicalGridView

Use the *AxHierarchicalGridView* control when you want to display hierarchical data in a grid format. For example, you might have a grid that displays a list of tasks in a project. With this control, each task can have subtasks and you can present all tasks and subtasks in a single grid, as shown in Figure 7-6.

Title	StartDate	EndDate
Task A	1/4/2011	2/10/2011
Task B	1/4/2011	1/29/2011
Task C	1/9/2011	2/10/2011
Task D	1/20/2011	5/1/2011
Task E	1/20/2011	3/2/2011
Task G	3/5/2011	4/12/2011
Task F	2/15/2011	4/12/2011

FIGURE 7-6 Example of an *AxHierarchicalGridView* control in the user interface.

You use the *HierarchyIdFieldName* property to uniquely identify a row and the *HierarchyParentId-FieldName* property to identify the parent of a row. The following example illustrates the markup for an *AxHierarchicalGridView* control:

```
<dynamics:AxDataSource ID="AxDataSource1" runat="server" DataSetName="Tasks"
ProviderView="Tasks">
</dynamics:AxDataSource>
<dynamics:AxHierarchicalGridView ID="AxHierarchicalGridView1" runat="server"
    BodyHeight="" DataKeyNames="RecId" DataMember="Tasks"
    DataSetCachingKey="e779ece0-43b7-4270-9dc9-33f4c61d42b7"
    DataSourceID="AxDataSource1" EnableModelValidation="True"
    HierarchyIdFieldName="TaskId" HierarchyParentIdFieldName="ParentTaskId">
    <Columns>
        <dynamics:AxBoundField DataField="Title" DataSet="Tasks"
            DataSetView="Tasks" SortExpression="Title">
        </dynamics:AxBoundField>
        <dynamics:AxBoundField DataField="StartDate" DataSet="Tasks"
            DataSetView="Tasks" SortExpression="StartDate">
        </dynamics:AxBoundField>
        <dynamics:AxBoundField DataField="EndDate" DataSet="Tasks"
            DataSetView="Tasks" SortExpression="EndDate">
        </dynamics:AxBoundField>
    </Columns>
</dynamics:AxHierarchicalGridView>
```

AxContextMenu

Use the *AxContextMenu* control to create and display a context menu. This control provides methods to add and remove menu items and separators at run time. It also provides methods to resolve client or Enterprise Portal URLs, as shown in the following example:

```
AxUrlMenuItem myUrlMenuItem = new AxUrlMenuItem("MyUrlMenuItem");
AxContextMenu myContextMenu = new AxContextMenu();
myContextMenu.AddMenuItemAt(0, myUrlMenuItem);
```

AxGridView uses *AxContextMenu* when the *ShowContextMenu* property is set to *true*. You can access the *AxContextMenu* object by using the syntax *AxGridview.ContextMenu*.

AxFilter

Use the *AxFilter* control to filter the data that is retrieved from a data source. This control sets a filter on an instance of a *DataSetView* object by using an instance of a *AxDataSourceView* object, which is responsible for keeping the data synchronized with the filter that is set by calling the *SetAsChanged* and *ExecuteQuery* methods when data has changed. *AxDataSourceView* and *DataSetView* expose the following properties that you can use to access the filter programmatically:

- ***SystemFilter*** Gets the complete list of ranges on the query, including open, hidden, and locked, into the *conditionCollection* property on the filter object.
- ***UserFilter*** Gets only the open ranges on the *QueryRun* property into the *conditionCollection* property on the filter object.
- ***ResetFilter*** Clears the filter set on the *QueryRun* property and thus resets the filter (ranges) set programmatically.

You can set the range in the dataset in X++ as follows:

```
qbrBlocked =  qbds.addRange(fieldnum(CustTable,Blocked));
qbrBlocked.value(queryValue(CustVendorBlocked::No));
qbrBlocked.status(RangeStatus::Hidden);
```

To read the filter that is set on the data source in a web user control, use one of the following lines of code:

```
this.AxDataSource1.GetDataSourceView(this.AxGridView1.DataMember).SystemFilter.ToXml();
```

or

```
this.AxDataSource1.GetDataSet().DataSetViews[this.AxGridView1.DataMember].SystemFilter.ToXml();
```

The return value will look something like the following:

```
<?xml version="1.0" encoding="utf-16"?><filter xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="CustTable"><condition
attribute="Blocked" operator="eq" value="No" status="hidden" /></filter>
```

You can also set the filter programmatically:

```
string myFilterXml = @"<filter name='CustTable'><condition attribute='CustGroup' status='open'
value='10' operator='eq' /></filter>";
this.AxDataSource1.GetDataSourceView(this.AxGridView1.DataMember).SystemFilter.
AddXml(myFilterXml);
```

The *AxGridView* control also uses *AxFilter* when *ShowFilter* is set to *true*. You can access the *AxFilter* object by using *AxGridview.FilterControl* and the filter XML by using *AxGridView.Filter*. The filter reads the metadata from the *AxDataSource* component that is linked to the grid and displays filtering controls dynamically so that the user can filter the data source on any of the fields that are not hidden or locked. The filtering controls are rendered above the grid.

AxLookup

Use the *AxLookup* control on data entry pages to help the user pick a valid value for a field that references keys from other tables. In Enterprise Portal, lookups are metadata-driven by default and are automatically enabled for fields based on the relationship defined by metadata in the AOT.

The Customer Group lookup on the Customer details page is an example of a lookup that is automatically enabled. The extended data type (EDT) and table relationship metadata in the AOT define a relationship between the Customer table and the Customer group table. A lookup is automatically rendered so that the user can choose a customer group in the Customer group field when creating a customer record. You don't need to write any code to enable this behavior—it happens automatically.

In some scenarios, the automatic behavior isn't sufficient, and you might be required to customize the lookup. The lookup infrastructure of Enterprise Portal offers flexibility and customization options in both X++ and in C#, so that you can tailor the lookup user interface and the data retrieval logic to meet your needs.

To control the lookup behavior, in the *Data Set* node in the AOT, you can override the *dataSetLookup* method of a field in the data source. For example, if you want to filter the values that are displayed, you override *dataSetLookup*, as shown in the following X++ code:

```
void dataSetLookup(SysDataSetLookup sysDataSetLookup)
{
    List                    list;
    Query                   query = new Query();
    QueryBuildDataSource    queryBuildDataSource;
    Args                    args;

    args = new Args();
    list = new List(Types::String);
    list.addEnd(fieldstr(HcmGoalHeading, GoalHeadingId));
    list.addEnd(fieldstr(HcmGoalHeading, Description));

    queryBuildDataSource = query.addDataSource(tablenum(HcmGoalHeading));

    queryBuildDataSource.addRange(fieldnum(HcmGoalHeading,Active)).value(
        queryValue(NoYes::Yes));
```

```
    sysDataSetLookup.parmLookupFields(list);
    sysDataSetLookup.parmSelectField(fieldStr(HcmGoalHeading,GoalHeadingId));

    // Pass the query to SysDataSetLookup so it result is rendered in the lookup page.
    sysDataSetLookup.parmQuery(query);
}
```

In the preceding example, the entire list is built dynamically and *addRange* is used to restrict the values. The *SysDataSetLookup* class in X++ provides many properties and methods to control the behavior of the lookup.

You can also customize the lookup in C# in the web user control by writing code in the *Lookup* event of bound fields or by using the *AxLookup* control for fields that don't have data binding. To use *AxLookup* to provide lookup values for any ASP.NET control that isn't data bound, set the *TargetControlID* property of *AxLookup* to the ASP.NET control to which the lookup value is to be returned. Alternatively, you can base *AxLookup* on the EDT, the dataset, the custom dataset, or the custom user control by specifying the *LookupType* property. You can also control which fields are displayed in the lookup and which ones are returned. You can do this either through the markup or through code. You can write code to override the *Lookup* event and control the lookup behavior, as shown in the following code:

```
protected void AxLookup1_Lookup(object sender, AxLookupEventArgs e)
    {
        AxLookup lookup = (AxLookup)sender;

        // Specify the lookup fields
        lookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            lookup.LookupDataSetViewMetadata.ViewFields["CustGroup"]));

        lookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            lookup.LookupDataSetViewMetadata.ViewFields["Name"]));
}
```

AxActionPane

The *AxActionPane* control performs a function similar to the Action pane web part. You can use it to display the Action pane at the top of the page, similar to the SharePoint ribbon. Use the *WebMenuName* property of the *AxActionPane* control to reference the web menu that contains the menu items to display on the Action pane as buttons. To improve discoverability, the buttons are displayed in tabs and groups. You can use the *DataSource* and *DataMember* properties of the *AxActionPane* control to associate the Action pane buttons with data.

To use the *AxActionPane* control in a web user control, you need to add a reference to the *Microsoft.Dynamics.Framework.Portal.SharePoint* assembly. You can do this by adding the following lines in the markup for the web user control:

```
<%@ Register Assembly="Microsoft.Dynamics.Framework.Portal.SharePoint, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" Namespace="Microsoft.Dynamics.Framework.
Portal.SharePoint.UI.WebControls" TagPrefix="dynamics" %>
```

If you prefer, you can use the Action pane web part as an alternative to the *AxActionPane* control.

AxToolbar

The *AxToolbar* control performs a function similar to the Toolbar web part. You can use it to display a toolbar at a certain location on the page instead of using the Action pane at the top of the page. For example, you might choose to display a toolbar at the top of a grid control with New, Edit, and Delete actions.

Internally, *AxToolbar* uses the SharePoint toolbar controls. *AxToolbarButton*, which is used within *AxToolbar*, is derived from *SPLinkButton*. It is used to render top-level buttons. Similarly, *AxToolBarMenu*, which is used within *AxToolbar*, is derived from *Microsoft.SharePoint.WebControls.Menu*. This control renders a drop-down menu by means of a callback when a user clicks a button. Thus, the menu item properties can be modified before the menu items are rendered.

If you prefer, you can use the Toolbar web part as an alternative to *AxToolbar*. Generally, you use the Toolbar web part to control the display of toolbar menu items. But if you have a task page that contains both master and detail information, such as a purchase requisition header and line items, you should use place *AxToolbar* in your web user control above the *AxGridview* control containing the details to allow the user to add and manage the line items.

You can bind *AxToolbar* to an *AxDataSource* or use it as an unbound control. When the controls are bound, the menu item context is automatically based on the item that is currently selected. When the controls are unbound, you must write code to manage the toolbar context.

You can point the toolbar to a web menu in the AOT by using the *WebMenuName* property. With a web menu, you can define a multilevel menu structure with the *SubMenu*, *MenuItem*, and *MenuItem* reference nodes. Each top-level menu item is rendered by using the *AxToolbarButton* control as a link button. Each top-level submenu is rendered by using the *AxToolbarMenu* control as a drop-down menu. If you have submenus, additional levels are displayed as submenus.

SetMenuItemProperties, *ActionMenuItemClicking*, and *ActionMenuItemClicked* are events that are specific to *AxToolbar*. You use *SetMenuItemProperties* to change the behavior of drop-down menus; for example, to show or hide menu items based on the currently selected record, set or remove context, and so on. The following code shows an example of how to change the menu item context in the *SetMenuItemProperties* event:

```
void Webpart_SetMenuItemProperties(object sender, SetMenuItemPropertiesEventArgs e)
{
    // Do not pass the currently selected customer record context,
    // since this menu is for creating new (query string should be empty)
    if (e.MenuItem.MenuItemAOTName == "EPCustTableCreate")
    {
        ((AxUrlMenuItem)e.MenuItem).MenuItemContext = null;
    }
}
```

If you have defined user interface logic in a web user control and want to call this function instead of the one defined in the AOT when a toolbar item is clicked, use *ActionMenuItemClicking*

and *ActionMenuItemClicked*. For example, you can prevent a menu item from executing the action defined in the AOT by using the *ActionMenuItemClicking* event and defining your own action in C# by using the *ActionMenuItemClicked* event in the web user control, as shown here:

```
void webpart_ActionMenuItemClicking(object sender, ActionMenuItemClickingEventArgs e)
{
    if (e.MenuItem.MenuItemAOTName.ToLower() == "EPCustTableDelete")
    {
        e.RunMenuItem = false;
    }
}

void webpart_ActionMenuItemClicked(object sender, ActionMenuItemEventArgs e)
{
    if (e.MenuItem.MenuItemAOTName.ToLower() == "EPCustTableDelete")
    {
        int selectedIndex = this.AxGridView1.SelectedIndex;

        if (selectedIndex != -1)
        {
            this.AxGridView1.DeleteRow(selectedIndex);
        }
    }
}
```

AxPopup controls

Use an *AxPopup* control to open a page in a pop-up browser window, to close a pop-up page, or to pass data from the pop-up page to the parent page and trigger a *PopupClosed* server event on the parent. This functionality is encapsulated in two controls: *AxPopupParentControl*, which you use on the parent page, and *AxPopupChildControl*, which you use on the pop-up page itself. Both controls are derived from *AxPopupBaseControl*. These controls are AJAX-compatible, so you can create them conditionally as part of a partial update.

AxPopupParentControl allows a page, typically a web part page, to open in a pop-up window. You can open a pop-up window from a client-side script by using the *GetOpenPopupEventReference* method. The string that is returned is a JavaScript statement that can be assigned, for example, to a button's *OnClick* attribute or to a toolbar menu item. The following code shows how to open a pop-up window with client-side scripting by modifying the *OnClick* event:

```
protected void SetPopupWindowToMenuItem(SetMenuItemPropertiesEventArgs e)
{
    AxUrlMenuItem menuItem = new AxUrlMenuItem("EPCustTableCreate");

    //Calling the JavaScript function to set the properties of opening web page
    //on clicking the menuitems.
    e.MenuItem.ClientOnClickScript =
        this.AxPopupParentControl1.GetOpenPopupEventReference(menuItem);
}
```

You can also open a pop-up window from a server method by calling the *OpenPopup* method. Because pop-up blockers can block server-initiated pop-up windows, use *OpenPopup* only when necessary.

When placed on a pop-up page, *AxPopupChildControl* allows the page to close. You can close the pop-up page with a client-side script by using the *GetClosePopupEventReference* method, as shown in the following example:

```
this.BtnOk.Attributes.Add("onclick",
    this.popupChild.GetClosePopupEventReference(true, true) + "; return false;");
```

You can close a pop-up window from the server event by using the *ClosePopup* method. Use the server method when additional processing is necessary upon closing, such as performing an action or calculating values to be passed to the parent page. The *ClosePopup* and *OpenPopup* methods have two parameters:

- ***setFieldValues*** When *true*, this indicates that data must be passed back to the parent page.
- ***updateParent*** When *true*, this indicates that the parent page must post back after the pop-up page is closed. *AxPopupChildControl* makes a call (through a client-side script) to the parent page to post back, with the *AxPopupParentControl* as the target. *AxPopupParentControl* then triggers the *PopupClosed* server event. When the event is triggered, the application code of the parent page can receive the values that are passed from the pop-up page and perform an action or update its state.

You can pass data from the pop-up page back to the parent page by using *AxPopupField* objects. You expose these objects through the *Fields* property of *AxPopupBaseControl*, from which both *AxPopupParentControl* and *AxPopupChildControl* are derived. *AxPopupParentControl* and *AxPopupChildControl* have fields with the same names. When the pop-up page closes, the value of each field of *AxPopupChildControl* is assigned (through a client-side script) to the corresponding field in *AxPopupParentControl*.

Optionally, you can associate *AxPopupField* with another control, such as *TextBox* (or any other control), by assigning the *TargetId* property of the *AxPopupField* control to the ID property of the target control. This is useful, for example, when the pop-up page has a *TextBox* control. To pass the user input to the parent page on closing the pop-up page—and to perform the action entirely on the client to avoid a round-trip—you need to associate a field with the *TextBox* control. When *AxPopupField* isn't explicitly associated with a target control, it is implicitly associated with a *HiddenField* control that is created automatically by *AxPopupParentControl* or *AxPopupChildControl*.

You can then set the value of the field on the server by using the *SetFieldValue* method. Typically, you call *SetFieldValue* on *AxPopupChildControl*, and you can call it at any point that the user interacts with the pop-up page, including the initial rendering or the closing of the page. You can retrieve the value of the field by using the *GetFieldValue* method. Typically, you call this method on *AxPopupParentControl* during the processing of the *PopupClosed* event. You can clear the values of non-associated fields by calling the *ClearFieldValues* method.

You can also set or retrieve values of *AxPopupFields* on the client by manipulating the target control value. You can retrieve target control, whether explicitly or implicitly associated, by using the *TargetControl* property.

BoundField controls

BoundField controls are used by data-bound controls (such as *AxGridView*, *AxGroup*, ASP.NET *GridView*, and ASP.NET *DetailsView*) to display the value of a field through data binding. The way in which a bound field control is displayed depends on the data-bound control in which it is used. For example, the *AxGridView* control displays a bound field control as a column, whereas the *AxGroup* control displays it as a row.

The Enterprise Portal framework provides a number of enhanced bound field controls that are derived from ASP.NET bound field controls but are integrated with the Microsoft Dynamics AX metadata. These controls are described in Table 7-2.

TABLE 7-2 Microsoft Dynamics AX *BoundField* controls.

Control	Description
AxBoundField	Used to display text values. The *DataSet*, *DataSetView*, and *DataField* properties define the source of the data.
AxHyperLinkBoundField	Used to display hyperlinks. Use the *MenuItem* property to point to a web menu item in the AOT for generating the URL and the *DataSet*, *DataSetView*, and *DataField* properties to define the source of the data. If the web menu name is stored within the record, use the *DataMenuItemField* property instead of *MenuItem*.
AxBoundFieldGroup	Used to display *FieldGroups* defined in the AOT. The *DataSet*, *DataSetView*, and *FieldGroup* properties define the source of the data.
AxCheckBoxBoundField	Used to display a Boolean field in a check box. The *DataSet*, *DataSetView*, and *DataField* properties define the source of the data.
AxDropDownBoundField	Used to display a list of values in a drop-down menu. The *DataSet*, *DataSetView*, and *DataField* properties define the source of the data.
AxRadioButtonBoundField	Used to display a list of values as radio buttons. The *DataSet*, *DataSetView*, and *DataField* properties define the source of the data. Use the *RepeatDirection* property to define whether the radio button should be rendered horizontally or vertically
AxReferenceBoundField	Used for a surrogate key. The surrogate key is typically an identifier to a row in another related table. Instead of directly displaying the surrogate key, the value of fields in the *AutoIdentification* field group of the related table is displayed. These are more readable and user friendly.

Depending on the field type, the Bound Field Designer in Visual Studio automatically groups fields under the correct bound field type.

AxContentPanel

The *AxContentPanel* control extends the ASP.NET *UpdatePanel* control. It acts as a container for other controls and allows for partial updates of the controls that are placed inside it, eliminating the need to refresh the entire page. It also provides a mechanism to provide and consume record context for its child controls.

AxPartContentArea

Use *AxPartContentArea* to define the FactBox area in a control. This control acts as a container for *AxInfoPart*, *AxFormPart*, and *CueGroupPartControl*.

AxInfoPart

Use the *AxInfoPart* control to display an Info Part. This control must be placed inside an *AxPartContentArea* control.

AxFormPart

Use the *AxFormPart* control to display a Form Part. This control must be placed inside an *AxPartContentArea* control.

CueGroupPartControl

Use the *CueGroupPartControl* control to display a Cue Group. This control must be placed inside an *AxPartContentArea* control.

AxDatePicker

Use the *AxDatePicker* control to a calendar control that allows a user to pick a date.

AxReportViewer

Use the *AxReportViewer* control to display an SSRS report.

Developing for Enterprise Portal

To develop Enterprise Portal applications, you use a combination of MorphX, Visual Studio, and SharePoint products and technologies:

- **MorphX** You use MorphX to develop the data and business tier components in your application. You also use MorphX to define navigation elements; store unified metadata and files; import and deploy controls, pages, and list definitions; and generate proxies. For more information about MorphX, see Chapter 2, "The MorphX development environment and tools."
- **Visual Studio** You use Visual Studio for developing and debugging web user controls. The Visual Studio Add-in for Enterprise Portal provides project and control templates to speed the development process. Visual Studio provides an easy way to add new controls to the AOT; tools for importing controls and style sheets from the AOT; and the capability to work with proxies. The Enterprise Portal framework provides various APIs for accessing data and metadata.
- **SharePoint products and technologies** You use SharePoint to develop web part pages and lists. You also use it to edit master pages, which contain the common elements for all the pages in a site. With a browser, you can use the Create or Edit Page tool of SharePoint to

design your web part page. You can also use SharePoint Designer to create or edit both web part pages and master pages.

The AOT controls all metadata for Enterprise Portal and stores all of the controls and pages that you develop in Visual Studio and SharePoint. It also stores other supporting files, definitions, and features under the *Web* node.

This section walks you through the steps necessary to create an Enterprise Portal list page and details page, and explains how to improve performance by using AJAX. For information about the Enterprise Portal user interface, see Chapter 5.

Create a model-driven list page

Microsoft Dynamics AX 2012 introduces a new model-driven way of creating list pages. With Microsoft Dynamics AX 2009, you had to create a form to be displayed in the Microsoft Dynamics AX client and a webpage to be displayed in Enterprise Portal. With model-driven list pages, you model the list page once and can have it appear in both the client and in Enterprise Portal. The form displayed in the Microsoft Dynamics AX client and the webpage displayed in Enterprise Portal share code and metadata. Any changes to the form are reflected automatically in both the client and in Enterprise Portal. This leads to a number of advantages, such as reduced development effort, a unified code base, and easier maintenance.

Figure 7-7 show an example of the development environment for creating a model-driven list page.

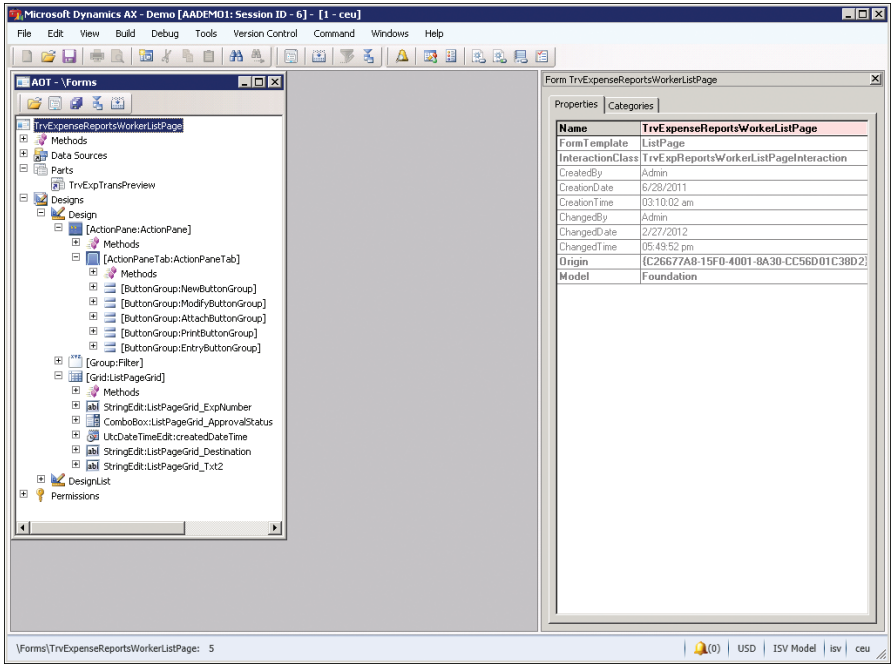


FIGURE 7-7 Model-driven list page development.

The following are high-level steps that you can follow to create a model-driven list page.

1. Start the Development Workspace.
2. Create a new form in the AOT and set the *FormTemplate* property to *ListPage*. This setting automatically adds design elements such as the filter, grid, and Action pane.
3. Set the query on the form to get the data that you want the form to display.
4. Set the *DataSource* property on the grid to the required data view.
5. Add the fields that you want to display in the grid.
6. Create and add an Action pane, and Info Parts if required. Ideally, you should create one Info Part to be displayed in the Preview Pane (below the grid) and one or more Info Parts, Form Parts, and Cue Groups to be displayed in the FactBox area (to the right of the grid). The Preview Pane should display extended information about the selected record and the FactBoxes should display related information. To link these parts to the list page, you will need to create the corresponding display menu items.
7. Create a display menu item that points to the form. Right-click the menu item, and click Deploy To EP.
8. When prompted, select the module that you want to deploy the page to. This will automatically create a SharePoint web part page for the list page for Enterprise Portal.
9. It will also create a URL web menu item and import the corresponding page definition in the AOT.
10. Set the *HyperLinkMenuItem* property on the first field in the grid to a display menu item that corresponds to a details page, and then refresh the AOD. This will render links in the first column that can be used to open up the record using a linked details page.

Define a list page interaction class

To achieve more control over how your model-driven list page behaves, you can specify a custom interaction class by using the *InteractionClass* property of the form. The name of your class should end with *ListPageInteraction* and can inherit either the *SysListPageInteractionBase* class, which is easy to use, or the *ListPageInteraction* class, which is more flexible.

The *SysListPageInteractionBase* class provides methods that you can override and serve as a place to put custom code. The following are some of these methods:

- ***initializing*** Called when the list page initializes.
- ***selectionChanged*** Called when the user selects a different record on the list page.
- ***setButtonEnabled*** Enables or disables buttons, called from the *selectionChanged* method.

- ***setButtonVisibility*** Displays or hides buttons. This method is called once when the form opens.
- ***setGridFieldVisibility*** Shows or hides grid fields. This method is called once when the form opens.

For more information, see the topic "*SysListPageInteractionBase* Class," at *http://msdn.microsoft.com/en-us/library/syslistpageinteractionbase.aspx*.

Create a details page

A details page in Enterprise Portal displays detailed information about a specific record.

Use the following high-level steps to create a details page:

1. In Visual Studio, use the EP Web Application Project template (under the Microsoft Dynamics AX category) to create a new project.
2. Add a new item to the project by using the EP User Control with the Form template (found under the Microsoft Dynamics AX category). This automatically adds the control to AOT.
3. Switch to design view, select the *AxDataSource* control (Figure 7-8), and then set the *DataSet* name.

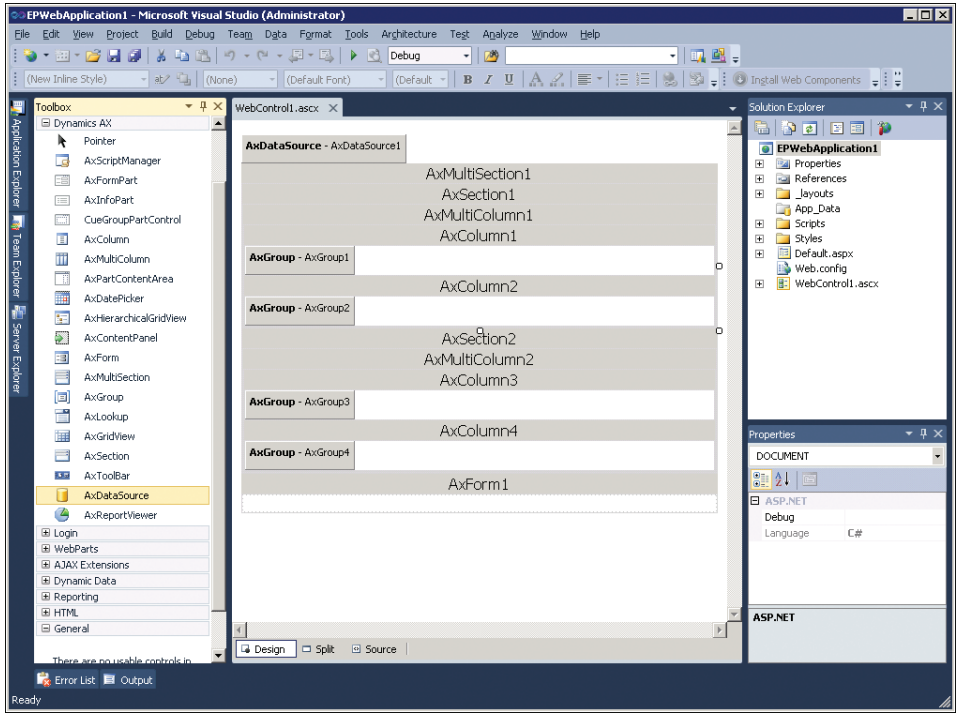


FIGURE 7-8 Creating a details page in Visual Studio.

4. Select the *AxForm* control, and then ensure that *DataSourceID* is set to the *AxDataSource*.
5. Set the *DataMember* and *DataKeyNames* on the form as appropriate.
6. If required, change the default mode of the form to *Edit* or *Insert* (it is *ReadOnly* by default).
7. To autogenerate the Save and Close buttons, do the following:
   - In *ReadOnly* mode, set *AutoGenerateCancelButton* to *true*.
   - In *Edit* mode, set *AutoGenerateEditButton* to *true*.
   - In *Insert* mode, set *AutoGenerateInsertButton* to *true*.
   - Select an *AxGroup* control and ensure that the *FormID* property is set.
8. Click the Edit Fields link and add the required fields to the *AxGroup* control.
9. Compile the EP Web Application by using the Build menu. Ensure that there are no errors. Compiling the application automatically deploys the control to the SharePoint directory.
10. In Microsoft Dynamics AX, start the Development Workspace, and then navigate to \Web\ Web Content\Managed.
11. Right-click the managed item that maps to the web user control that you created, and then click Deploy To EP.
12. When prompted, select the module you want to deploy the page to. This will automatically create a SharePoint web part page for Enterprise Portal and put your web user control on the page using the User control web part. It will also create a URL web menu item and import the corresponding Page Definition in AOT.
13. Select the web menu item created for the page, and then set *WindowMode* to *Modal*. This will cause the details page to open in a modal dialog box.
14. Create a new display menu item and set the *WebMenuItemName* property to the web menu item that is linked to the details page.
15. Use this display menu item to link to the details page from the list page grid, as described in the "Create model-driven list page" section earlier in this chapter.

Modal dialog settings

Enterprise Portal uses modal dialogs to implement standard interaction patterns for pages. In Microsoft Dynamics AX 2012, Enterprise Portal includes two new metadata settings, *WindowMode* and *WindowSize*, on the web menu item, which that you can use to implement these interaction patterns without writing any code.

WindowMode has the following four settings:

- ***Inline*** Causes the target URL to open in the same window. This setting replaces the current page with the target page that the menu item links to.

- ***Modal*** Causes the target URL to open in a modal dialog on top of the window. The current page is still available in the background. However, because the dialog is modal, the user can interact only with the modal dialog and not with the page that is in the background.

  If a web menu item with *WindowMode* set to *Modal* is opened from within a modal dialog, the modal dialog is reused. The page currently open in the modal dialog is replaced with the target page that the menu item links to.

- ***NewModal*** Functions in a similar way to the *Modal* setting but does not reuse an existing modal dialog. Therefore, if a web menu item with *WindowMode* set to *NewModal* is opened from within a modal dialog, a second-level modal dialog opens on top of the old one (as shown in Figure 7-9).

- ***NewWindow*** Causes the target URL to open in a new window.

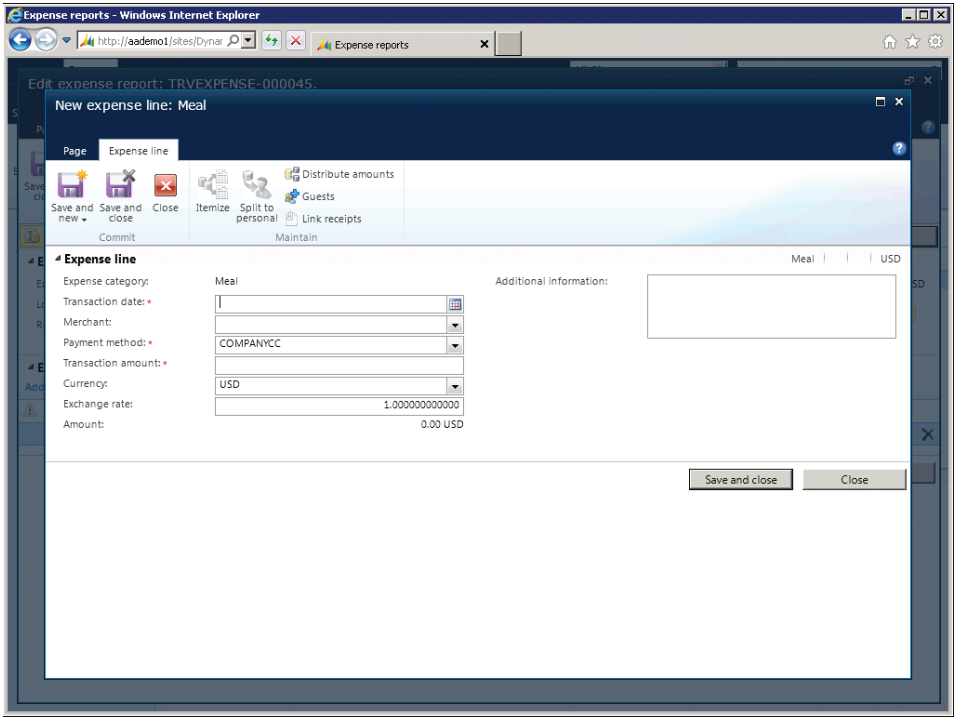


FIGURE 7-9 Example of an Enterprise Portal page with two levels of modal dialogs.

WindowSize has five settings: *Maximum*, *Large*, *Medium*, *Small*, and *Smallest*. These settings correspond to five predefined sizes for the modal dialogs.

AJAX

You can use .NET AJAX to create ASP.NET webpages that can update data on the page without refreshing the entire page. AJAX provides client-side and server-side components that use the *XMLHttpRequest* object, along with JavaScript and DHTML, to enable portions of the page to update asynchronously, again without refreshing the entire page. With AJAX, you can develop Enterprise Portal webpages just as you would any regular ASP.NET page, and you can declaratively mark the components that should be rendered asynchronously.

By using the *UpdatePanel* server control, you can enable sections of a webpage to be partially rendered without an entire page postback. The User control web part contains the *UpdatePanel* server control internally. The script library is included in the master page, so that any control can use AJAX without the need to write any explicit markup or code.

For example, if you add a text box and button and write code for the button's click event on the server without AJAX, when a user clicks the button, the entire page is refreshed. But when you load the same control through the User control web part, as in the following example, the button uses AJAX and updates the text box without refreshing the entire page:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" />
```

In the code-behind, update the text box with the current time after 5 seconds:

```
protected void Button1_Click(object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(5000);
    TextBox1.Text = System.DateTime.Now.ToShortTimeString();
}
```

If you want to override the AJAX behavior and force a full postback, you can use the *PostBackTrigger* control in the User control, as shown here:

```
<%@ Register assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" Namespace="System.Web.UI" TagPrefix="asp" %>

<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" />
    </ContentTemplate>
    <Triggers>
        <asp:PostBackTrigger ControlID="Button1" />
    </Triggers>
</asp:UpdatePanel>
```

Session disposal and caching

All web parts on a webpage share the same session in Microsoft Dynamics AX. After the page is served, the session is disposed of. To optimize performance, you can control the timeframe for the disposal of the session. Through settings in the Web.config file, you can specify the session timeout, in addition to the maximum number of cached concurrent sessions.

For example, to set the maximum number of cached concurrent sessions to 300 and the session timeout to 45 seconds, add the *<Microsoft.Dynamics>* section, as shown in the following example, after the *</system.web>* element. Remember that an increase in any of these values comes at the cost of additional memory consumption.

```
<Microsoft.Dynamics>
    <Session MaxSessions="300" Timeout="15" />
</Microsoft.Dynamics>
```

Many of the methods that you use in the Enterprise Portal framework to add code to a User control require access to the *Session* object. You also need to pass the *Session* object when using proxy classes. You can access the *Session* object through the web part that hosts the User control, as shown here:

```
AxBaseWebPart webpart = AxBaseWebPart.GetWebpart(this);
return webpart == null ? null : webpart.Session;
```

By default, Enterprise Portal uses the ASP.NET session state. However, you can configure and use Windows Server AppFabric distributed caching with Enterprise Portal to further improve performance in server farm environments. After you install and configure Windows Server AppFabric, you can specify the name and region for Enterprise Portal to use in the Web.config file.

For example, to set the cache name as *MyCache* and the region as *MyRegion*, add the *<Microsoft.Dynamics>* section, as shown here, after the *</system.web>* element in the Web.config file for Enterprise Portal:

```
<Microsoft.Dynamics>
    <AppFabricCaching CacheName="MyCache" Region="MyRegion" />
</Microsoft.Dynamics>
```

Context

Context is a data structure that is used to share data related to the current environment and user actions taking place with different parts of a web application. Context passes information to a web part about actions taking place in another control so that the web part can react. Context can also be used to pass information to a new page. Generally, information about the current record that the user is working on provides the information for the context. For example, when the user selects a row in a grid view, other controls might require information about the newly selected row so that they can react.

AxContext is an abstract class that encapsulates the concept of the context. The classes *AxTableContext* and *AxViewContext* derive from and implement *AxContext*. *AxTableContext* is for table-based context, and *AxViewContext* is for dataset view context. A view can contain more than one table, so it contains an *AxTableContext* object for each table in the view in the *TableContextList* collection. The *RootTableContext* property returns the *TableContext* of the root table in that dataset view. *AxViewDataKey* uniquely identifies the *AxViewContext*, and it contains the *TableDataKeys* collection. *AxTableDataKey* uniquely identifies *AxTableContext*. An event is raised whenever the context changes. If the context is changed within a User control, the *CurrentContextChanged* event is raised. If the context changes in other web parts that are connected to the User control, the *ExternalContextChanged* event is raised.

You can write code in these events on the *AxBaseWebPart* from your web user control and use the *CurrentContextProviderView* or *ExternalContextProviderView* and *ExternalRecord* properties to get the record associated with the context. You can trigger all of these events programmatically from your application logic by calling *FireCurrentContextChanged* or *FireExternalContextChanged* so that all other connected controls can react to the change that you made through your code. The following example triggers the *CurrentContextChanged* event:

```
void CurrentContextProviderView_ListChanged(object sender,
    System.ComponentModel.ListChangedEventArgs e)
{
    /* The current row (which is the current context) has changed update the consumer webparts.
       Fire the current context change event to refresh (re-execute the query) the consumer web
parts
    */
    AxBaseWebPart webpart = this.WebPart;
    webpart.FireCurrentContextChanged();
}
```

The following example gets the record from the connected web part.

First, subscribe to the *ExternalContextChanged* event in the consumer web user control, as shown here:

```
protected void Page_Load(object sender, EventArgs e)
{
    //Add Event handler for the ExternalContextChange event.
    //Whenever selecting the grid of the provider web part changes, this event gets fired.
    (AxBaseWebPart.GetWebpart(this)).ExternalContextChanged +=
        new
EventHandler<Microsoft.Dynamics.Framework.Portal.UI.AxExternalContextChangedEventArgs>
            (AxContextConsumer_ExternalContextChanged);
}
```

Next, get the record passed through the external context, as shown in the following example:

```
void AxContextConsumer_ExternalContextChanged(object sender,
    Microsoft.Dynamics.Framework.Portal.UI.AxExternalContextChangedEventArgs e)
{
    //Get the AxTableContext from the ExternalContext passed through web part connection and
    //construct the record object and get to the value of the fields
```

```
    IAxaptaRecordAdapter currentRecord = (AxBaseWebPart.GetWebpart(this)).ExternalRecord;

    if (currentRecord != null)
    {
        lblCustomer.Text = (string)currentRecord.GetField("Name");
    }
}
```

Data

The ASP.NET controls access and manipulate data through data binding to *AxDataSource*. You can also access the data through the APIs directly. The *Microsoft.Dynamics.AX.Framework.Portal.Data* namespace contains several classes that work together to retrieve data.

For example, use the following code to get the current row from the *DataSetView*:

```
private DataSetViewRow CurrentRow
{
    get
    {
        try
        {
            DataSetView dsv =
                this.ContactInfoDS.GetDataSet().DataSetViews[this.ContactInfoGrid.DataMember];

            return (dsv == null) ? null : dsv.GetCurrent();
        }
        // CurrentRow on the dataset throws exception in empty data scenarios
        catch (System.Exception)
        {
            return null;
        }
    }
}
```

To set the menu item with context for the current record, use the following code:

```
DataSetViewRow currentContact =
    this.dsEPVendTableInfo.GetDataSourceView(gridContacts.DataMember).DataSetView.GetCurrent();

using (IAxaptaRecordAdapter contactPersonRecord = currentContact.GetRecord())
{
    ((AxUrlMenuItem)e.MenuItem).MenuItemContext =
        AxTableContext.Create(AxTableDataKey.Create(
            this.BaseWebpart.Session, contactPersonRecord, null));
}
```

Metadata

The Enterprise Portal framework provides a rich set of APIs for accessing the metadata in the AOT through managed code. The *Microsoft.Dynamics.AX.Framework.Services.Client* namespace contains several classes that work together to retrieve metadata from the AOT. Enterprise Portal controls use

the metadata to retrieve information about formatting, validation, and security, among other things, and apply it in the user interface automatically. You can also use these APIs to retrieve the metadata and use it in your user interface logic.

The MetadataCache class is the main entry point for accessing metadata and provides static methods for this purpose. For example, to get the metadata for an enum, you use the *EnumMetadata* class and the *MetadataCache.GetEnumMetadata* method, as shown here:

```
/// <summary>
/// Loads the drop-down list with the enum values.
/// </summary>
private void LoadDropdownList()
{
    EnumMetadata salesUpdateEnum = MetadataCache.GetEnumMetadata(
        this.AxSession, EnumMetadata.EnumNum(this.AxSession, "SalesUpdate"));

    foreach (EnumEntryMetadata entry in salesUpdateEnum.EnumEntries)
    {
        ddlSelectionUpdate.Items.Add(new ListItem(
            entry.GetLabel(this.AxSession), entry.Value.ToString()));
    }
}
```

To get the label value for a table field, use the following code:

```
TableMetadata tableSalesQuotationBasketLine =
    MetadataCache.GetTableMetadata(this.AxSession, "CustTable");

TableFieldMetadata fieldItemMetadata = tableSalesQuotationBasketLine.
FindDataField("AccountNum");

String s = fieldItemMetadata.GetLabel(this.AxSession);
```

Figure 7-10 shows a portion of the object access hierarchy for metadata. For simplicity, not all APIs are included in the figure.

Proxy classes

If you need to access X++ classes, call table methods, or use enums in your user control, the Enterprise Portal framework provides an easy way of creating managed wrappers for these X++ objects. A proxy file internally wraps the .NET Business Connector calls and provides a simple, typed interface for C# applications.

Several predefined proxies are available for use in Enterprise Portal. They are defined in the EPApplicationProxies and the EPApplicationProxies1 projects in the AOT, which are located under \ Visual Studio Projects\C Sharp Projects. To use these proxy projects, open your web application project in Visual Studio, and then add a reference to these projects by clicking Project > Add EP Proxy Project. Then, in the web control, add a *using* statement to provide access to the proxy namespace, as shown here:

```
using Microsoft.Dynamics.Portal.Application.Proxy;
```

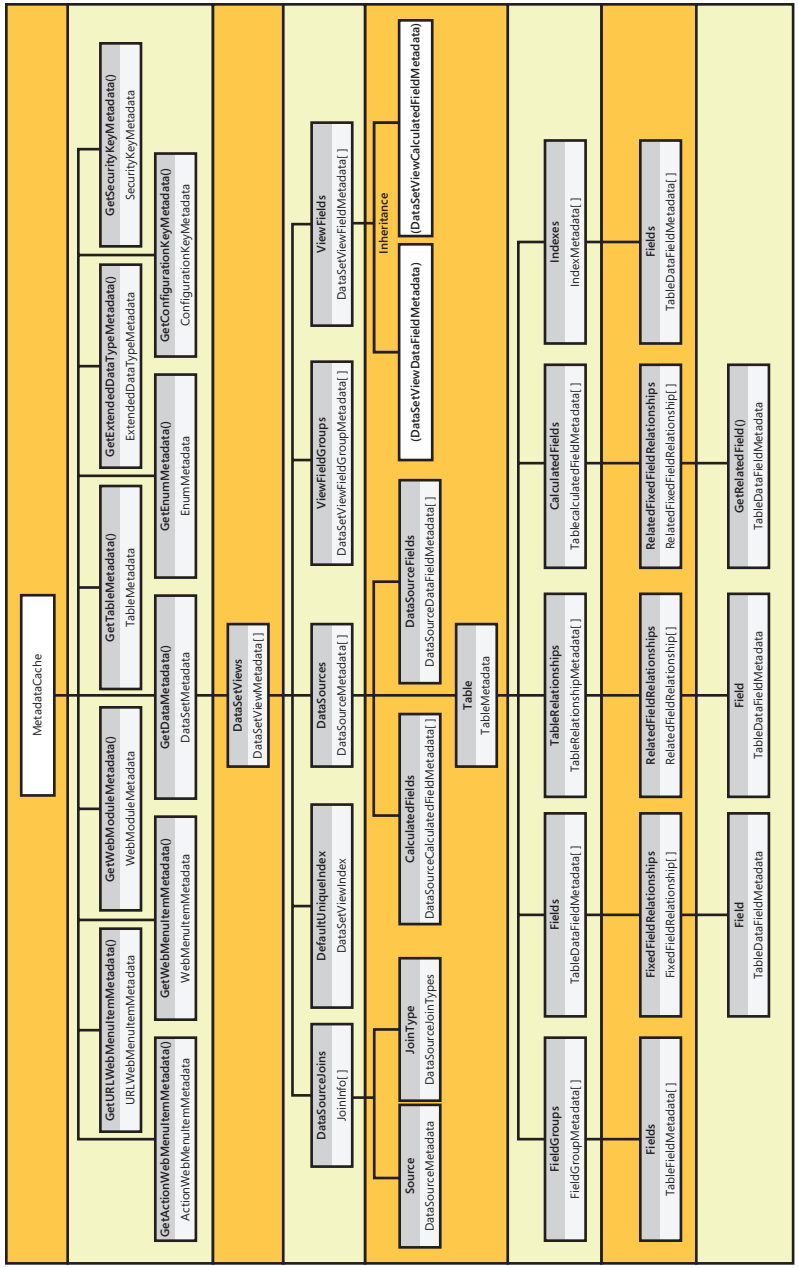


FIGURE 7-10 Metadata object hierarchy.

If you need to create a new proxy, you can create your own Visual C# class library project in Visual Studio by doing the following:

1. Set the default namespace of the project to *Microsoft.Dynamics.Portal.Application.Proxy*.

2. On the File menu, select the option to add the project to the AOT.

3. In Project Properties, set the *Deploy to EP* property to *Proxies*.

You can then add the objects from Application Explorer to the project, and the Enterprise Portal framework will automatically generate and deploy proxies for these to the App_Code folder of the IIS website. After you add a reference to a proxy project, you can access the X++ methods as though they are written in C#, as shown in Figure 7-11.

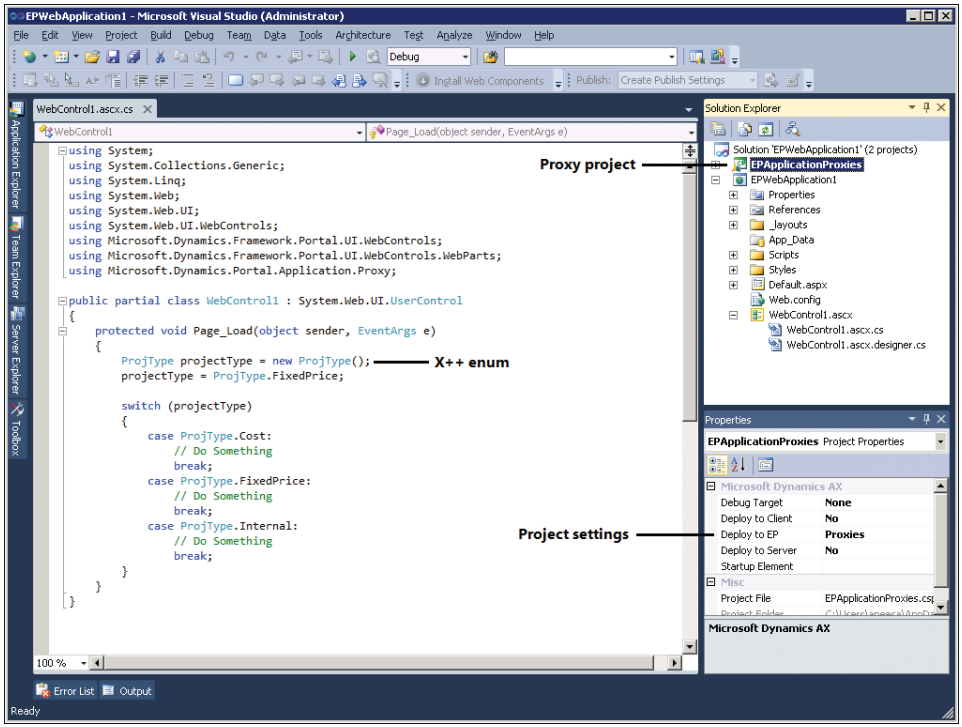


FIGURE 7-11 Working with proxies in Visual Studio.

ViewState

The web is stateless, which means that each request for a page is treated as a new request, and no information is shared. When loaded, each ASP.NET page goes through a regular page lifecycle, from initialization and page load onward. When a user interacts with the page, requiring the server to process control events, ASP.NET posts the values in the form to the same page to process the event on the server. A new instance of the webpage class is created each time the page is requested from the server. When postback happens, ASP.NET uses the *ViewState* feature to preserve the state of the page and controls so that changes made to the page during the round trip are not lost. The Enterprise Portal framework uses this feature, and Enterprise Portal ASP.NET controls automatically save their state to *ViewState*. The ASP.NET page reads the *ViewState* and reinstates the page and control state

during the regular page lifecycle. Therefore, you don't need to write any code to manage state if you're using Enterprise Portal controls. However, if you want to persist in-memory variables, you can write code to add or remove items from the *StateBag* class in ASP.NET, as shown here:

```
public int Counter
{
    get
    {
        Object counterObject = ViewState["Counter"];

        if (counterObject == null)
        {
            return 0;
        }

       return (int)counterObject;
    }

    set
    {
        ViewState["Counter"] = value;
    }
}
```

If you need to save the state of an X++ dataset, you can use the *pack-unpack* design pattern to store the state. For more information, see the topic "Pack-Unpack Design Pattern," at *http://msdn.microsoft.com/en-us/library/aa879675.aspx*.

The Enterprise Portal framework uses the ASP.NET *ViewState* property to store the state of most controls.

Labels

Microsoft Dynamics AX uses a localizable text resource file, the label file, to store messages that are displayed to the user. The label file is also used for user interface text, help text in the status bar, and captions. You can use labels to specify the user interface text in web controls and for element properties in the AOT *Web* node. You can add labels by setting the *Label* property in the AOT or by using X++ code.

When you use data-bound controls such as *AxGridView* or *AxForm* for the user interface, the bound fields automatically use the label associated with the field in the AOT and render it in the user's language at run time.

If you want to show a label in your web control for non-data-bound scenarios, use the *AxLabel* expression. *AxLabel* is a standard ASP.NET expression that looks up the labels defined in the AOT and renders them in the user's language when the page is rendered. To add the *AxLabel* expression, you can use the expression editor available in the design view of the web control by clicking the button that appears on the *(Expressions)* property. Alternatively, you can type the expression directly in the markup:

```
<asp:Button runat="server" ID="ButtonChange" Text="<%$ AxLabel:@SYS70959 %>"
OnClick="ButtonChange_Click" />
```

You can also add labels through code by using the *Labels* class, as shown here:

```
string s = Microsoft.Dynamics.Framework.Portal.UI.Labels.GetLabel("@SYS111587");
```

For better performance, Enterprise Portal caches the labels for all supported languages. If you add or change a label in the AOT, you need to clear the cache on the Enterprise Portal site by using the Refresh AOD command under Administration on the Enterprise Portal Home page.

Formatting

Microsoft Dynamics AX is a global product that supports multiple languages and is used in many countries/regions. Displaying data in the correct format for each localized version is a critical requirement for any global product. Through metadata, the Enterprise Portal framework recognizes the user's current locale and system settings to display data automatically in the correct format in data-bound controls.

If you're not using data-bound controls and want your unbound ASP.NET controls to be formatted like Enterprise Portal controls, you can use the *AxValueFormatter* class in the Enterprise Portal framework. This class implements the *ICustomFormatter* and *IFormatProvider* interfaces and defines a method that supports custom, user-defined formatting of an object's value. This method also provides a mechanism for retrieving an object to control formatting. For the various data types, specific *ValueFormatter* classes that are derived from *AxValueFormatter* are implemented: *AxStringValueFormatter*, *AxDateValueFormatter*, *AxDateTimeValueFormatter*, *AxTimeValueFormatter*, *AxRealValueFormatter*, *AxNumberValueFormatter*, *AxGuidValueFormatter*, and *AxEnumValueFormatter*.

You use *AxValueFormatterFactory* to create *AxValueFormatter* objects. You can create any of the preceding formatters, or you can create a formatter based on an EDT in Microsoft Dynamics AX. The data type for the extended data is retrieved from the metadata object for the EDT, and the culture information comes from the context. The various rules for languages and countries, such as number formats, currency symbols, and sort orders, are aggregated into a number of standard cultures. The Enterprise Portal framework identifies the culture based on the user's language setting in Microsoft Dynamics AX and makes this information available in the context. Formatter objects have a *Parse* method that you can use to convert a string value back into the underlying data type. For example, the following code formats the data based on a given EDT:

```
private string ToEDTFormattedString(object data, string edtDataType)
{
    ExtendedDataTypeMetadata edtType = MetadataCache.GetExtendedDataTypeMetadata(
        this.AxSession, ExtendedDataTypeMetadata.TypeNum(this.AxSession, edtDataType));

    IAxContext context = AxContextHelper.FindIAxContext(this);

    AxValueFormatter valueFormatter = AxValueFormatterFactory.CreateFormatter(
        this.AxSession, edtType, context.CultureInfo);

    return valueFormatter.FormatValue(data);
}
```

Validation

You use ASP.NET validator controls to validate user input on the server and, optionally, on the client (the browser). The Enterprise Portal framework includes ASP.NET validators that are specific to Microsoft Dynamics AX. *AxBaseValidator* derives from *System.Web.UI.WebControls.BaseValidator*, and *AxValueFormatValidator* derives from *AxBaseValidator*. Both are metadata-driven and are used intrinsically by bound fields. You can also use them in unbound scenarios.

ASP.NET validators are triggered automatically when a postback occurs that causes validation. For example, an ASP.NET button control causes validation on the client and the server when clicked. All validators that are registered on the page are validated. If a validator is found to be invalid, the page becomes invalid, and the *Page.IsValid* property returns a value of *false*.

The importance of *Page.IsValid* is best highlighted with an example. Suppose you add an ASP.NET button that executes some business logic in the *OnClick* event before redirecting the user to a different page. As mentioned earlier, the button causes validation by default, so validators are executed before the *OnClick* event is triggered. If you don't check to determine whether the page is valid in the *OnClick* event handler, the user is redirected even if a validation error occurs that requires the user's attention.

Enterprise Portal controls such as *AxForm* and *AxGridView* automatically check validation and won't perform the requested action if validation fails. The Microsoft Dynamics AX validator controls automatically write any validation errors to the Infolog.

When you're using ASP.NET controls directly instead of Enterprise Portal controls, as a best practice, make sure that your code examines the *Page.IsValid* property before any actions, such as navigating away from the current page, are completed. If errors occur, you'll want to keep the current page with Infolog displaying the errors so that the user will notice the errors and take corrective action.

Error handling

In Enterprise Portal, the .NET Business Connector (including proxies), the metadata, and the data layer all throw exceptions when error conditions occur. The Enterprise Portal ASP.NET controls automatically handle these exceptions, taking appropriate actions and displaying the errors in an Infolog.

Exceptions in Enterprise Portal are divided into three categories. These exception categories are defined in the *AxExceptionCategory* enumeration:

- ***NonFatal*** Indicates that the exception handling code should respond appropriately and allow the request to continue normally.
- ***AxFatal*** Indicates that an unrecoverable error has occurred in Enterprise Portal, and Enterprise Portal content will not be displayed. Content not related to Enterprise Portal should be displayed as expected.

- ***SystemFatal*** Indicates that a serious error, such as out of memory, has occurred and the request must be cancelled. Errors of this kind often cause an HTTP error code of 500.

Your code must handle any exceptions that might occur, if your code does any of the following:

- Directly calls methods in data layers from Enterprise Portal
- Directly calls metadata methods
- Uses proxy classes to call X++ methods

The following code shows how to use *AxControlExceptionHandler* in the *try-catch* statement to handle exceptions:

```
try
{
    // Code that may encounter exceptions goes here.
}
catch (System.Exception ex)
{
    AxExceptionCategory exceptionCategory;

    // Determine whether the exception can be handled.
    if (AxControlExceptionHandler.TryHandleException(this, ex, out exceptionCategory) == false)
    {
        // The exception was fatal and cannot be handled. Rethrow it.
        throw;
    }
    if (exceptionCategory == AxExceptionCategory.NonFatal)
    {
        // Application code to properly respond to the exception goes here.
    }
}
```

AxControlExceptionHandler tries to handle Microsoft Dynamics AX exceptions based on the three exception categories described earlier in this section. It returns a value of *true* if the type of exception is *NonFatal*.

Security

In Enterprise Portal, Microsoft Dynamics AX security is layered on top of, and depends on, the security of the underlying products and technologies, such as SharePoint and IIS. For external facing sites, communication security and firewall configurations are also important to help secure Enterprise Portal.

Enterprise Portal has two configurations in its site definition. The first, referred to as Microsoft Dynamics Public, allows Internet customers or prospective customers to view product catalogs, request customer accounts, and so on. The second, referred to as Microsoft Dynamics Enterprise Portal, is a complete portal for self-service scenarios involving intranet or extranet users who are authenticated employees, vendors, and customers.

The Microsoft Dynamics Public configuration has anonymous authentication enabled in both IIS and SharePoint so that anyone on the web can access it. To connect to Microsoft Dynamics AX, this configuration uses a built-in Microsoft Dynamics AX user account named Guest. The Guest account is part of the Enterprise Portal Guest user group, which has limited access to the Microsoft Dynamics AX components that are necessary for the public site to function.

The Microsoft Dynamics Enterprise Portal configuration uses either Integrated Windows authentication or Basic authentication over Secure Sockets Layer (SSL) that is enabled in IIS and SharePoint. This secured site restricts access to users with Active Directory accounts who are also configured as Microsoft Dynamics AX users and have access that has been enabled for the site by the Microsoft Dynamics AX system administrator. You use the System Administration > Setup > Users > User Relations dialog box in the Microsoft Dynamics AX client to set up users as an employee, vendor, business relation, or customer contact. Then you can grant them access to Enterprise Portal sites through Site groups for that Enterprise Portal site.

Both types of Enterprise Portal sites use the .NET Business Connector proxy account to establish connections to the AOS. The SharePoint application pool must be configured with a Windows domain user account, and this account must be specified as the Microsoft Dynamics AX .NET Business Connector proxy account for both sites to function. After the connection is established, Enterprise Portal uses either *LogonAsGuest* or *LogonAs*—depending on the type of Enterprise Portal site the current user has access to—to activate the Microsoft Dynamics AX security mechanism. Microsoft Dynamics AX provides various means and methods of limiting user access, such as placing restrictions on individual tables and fields and limiting the availability of application features through configuration keys and web configuration keys, as shown in Figure 7-12. User-level security can also be applied by using roles, duties, and privileges.

Enterprise Portal security is role based. This means that you can easily group tasks associated with a business function into a role, such as Sales or Consultant, and assign users to this role to give them the necessary permissions on the Microsoft Dynamics AX objects to perform those tasks in Enterprise Portal. To allow users access to more functionality, you can assign them to more than one role. For more information about roles, see Chapter 11, "Security, licensing, and configuration."

Secure web elements

To securely expose web controls through web parts in SharePoint, you can use privileges. You can either create a new privilege or use an existing one. You can add managed web content (Web\Web Content\Managed) or web menu items that reference URLs (Web\Web Menu Items\URLs) or actions (Web\Web Menu Items\Actions) as entry points for privileges to control which users can access them.

Remember to secure both the web menu item and the managed web content. If you only secure the web menu item (Figure 7-13), the user can still access the managed web content (for example, a web control) and can add it to a page that he or she has access to.

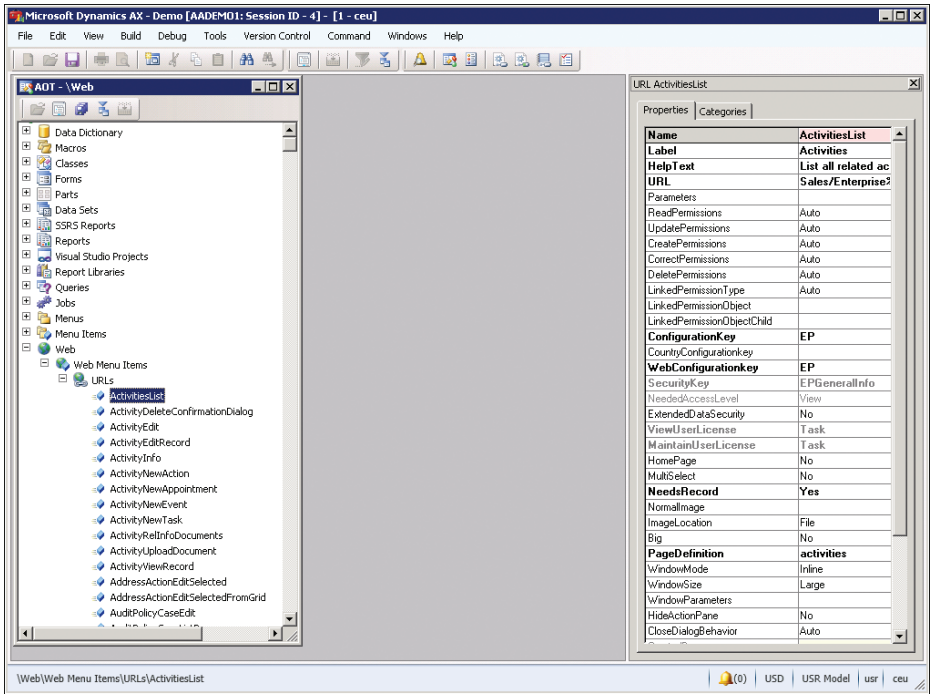


FIGURE 7-12 Assigning a configuration key and web configuration key to a web menu item.

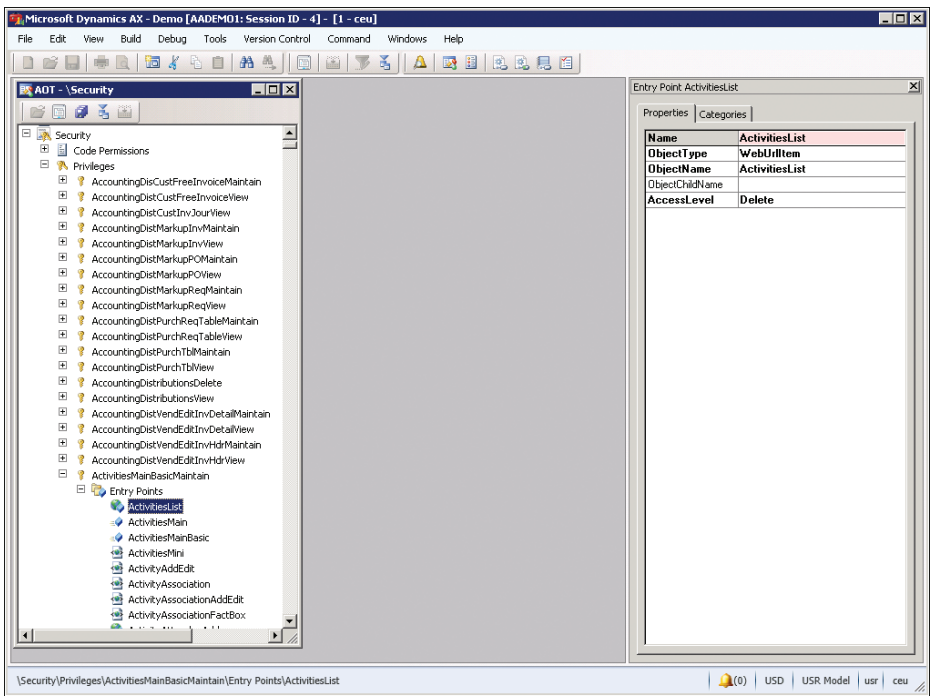


FIGURE 7-13 Adding a web menu item that references a URL as an entry point in a privilege.

At logon, the user's role determines the access. If a user doesn't have access to a web menu item, that item doesn't appear on the user's web menu. If a link in the web menu item appears in other web user controls that the user has access to, the item linked with the web menu item appears as text rather than a link.

If the user doesn't have access to web content on a webpage, the content isn't rendered on the page. Web part properties also limit the items that are displayed in the drop-down list based on the user permissions for the underlying objects. Moreover, the types of operations that are allowed on these objects depend on the access level set for the objects in the roles that the user belongs to.

Record context and encryption

Record context is the interface for passing information through the query string to a web part page to retrieve a record from Microsoft Dynamics AX. Enterprise Portal uses record context to locate a record in the Microsoft Dynamics AX database and display it in a web form for viewing and editing.

The following are some of the query string parameters that are used to pass the record context to an Enterprise Portal web part page:

- ***WTID*** Equals the Table ID
- ***WREC*** Equals the Rec ID
- ***WKEY*** Equals the Unique Record Key (the field identifier and the value of the field for the record to be retrieved)

These parameters are passed either in a query string or in post data on webpages. To help secure Enterprise Portal, Microsoft Dynamics AX uses a hash parameter. This ensures that a URL that is generated for one user cannot be used by any other user. For debugging and web development, the system administrator can turn off the encryption (use of the hash parameter) on the Enterprise Portal General tab of the Web Sites form, which is located in System Administration > Setup > Enterprise Portal > Web Sites. If record-level security and other data-level security are already active and no security threats exist, turning off the encryption could result in better performance. However, it is strongly recommended that you keep the encryption turned on.

SharePoint integration

Enterprise Portal is built on the SharePoint platform and takes advantage of some of the useful features and functionality offered by SharePoint to enable collaboration and content management.

Site navigation

The Enterprise Portal site uses the SharePoint navigation elements and object model for showing Microsoft Dynamics AX navigation items from the AOT. To display web menus from the AOT as the top and left navigation elements on the SharePoint site, Enterprise Portal setup adds the navigation

providers *DynamicsLeftNavProvider* and *DynamicsTopNavProvider*. For SharePoint Standard and Enterprise editions, *DynamicsMOSSTopNavProvider* is added instead of *DynamicsTopNavProvider*. The navigation providers override the default *TopNavigationDataSource* and *QuickLaunchDataSource*.

Web modules define the SharePoint sites and subsites in Enterprise Portal (for example, Sales, Employee Services, and so on). These are also used to build the top navigation bar. If you want to hide a link to a module from the top navigation bar, you can set the *ShowLink* property to *No* on the web module.

Web menus represent a collection of URL and action web menu items. You can use these elements to define the Quick launch structure for a web module, by setting the *QuickLaunch* property of a web module (see Figure 7-14) to the corresponding web menu. Alternatively you can use the Quick launch web part for this purpose. The links are automatically hidden or displayed based on the user's permissions. Web menu items help you create sites that are dynamic and versatile.

You can also use web menus with the Left navigation web part to provide navigation links on a page or web user control.

Internally, the framework uses the *WebLink* class to generate hyperlinks. This class has all the properties and methods that the framework needs to pass information back and forth between the browser and the server. More important, it has a method that returns the URL for the link. *WebLink* also has several methods for passing record information.

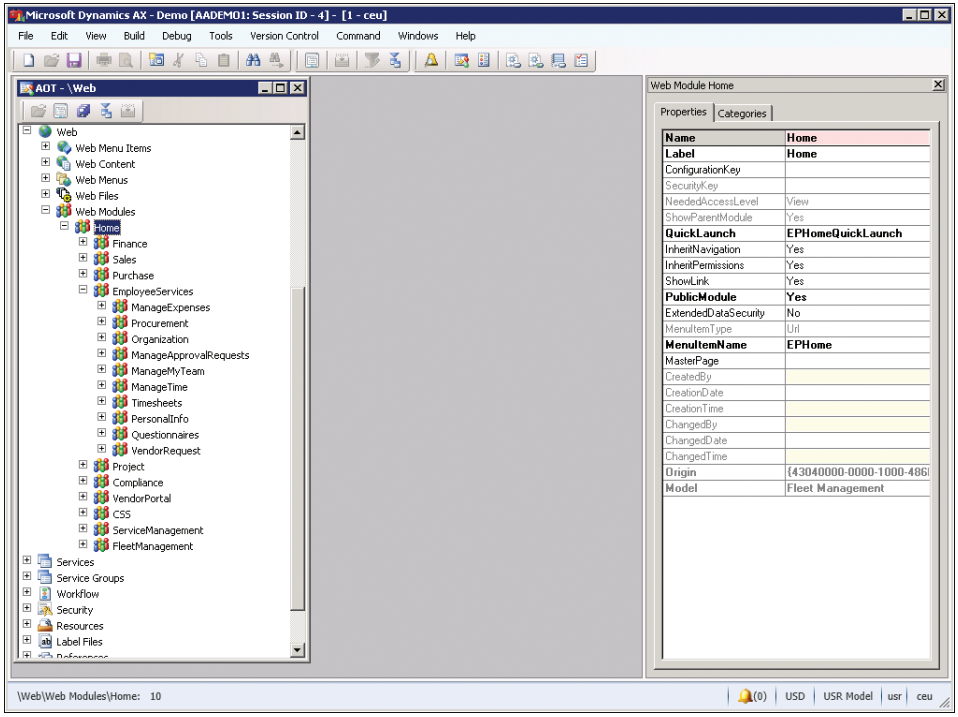


FIGURE 7-14 Web modules determine the sites, subsites, and Quick launch links that are displayed on a page.

Site definitions, page templates, and web parts

You can customize SharePoint sites by using site definitions or custom templates that are built on existing site definitions. Site definitions encompass multiple files that are located in the file system on each web server. These files define the structure and schema for the site. You can create new site definitions by copying the existing site definition files and modifying them to meet the needs of the new sites. You create custom templates by using the user interface to customize existing sites and storing them as templates.

The Enterprise Portal site definition files and custom templates are stored in the AOT under Web\Web Files\Static Files. Enterprise Portal setup deploys these files from the AOT to the web server file system and SharePoint.

Enterprise Portal includes one default site definition, which has two configurations: one for authenticated users and another for public Internet users. The site definition is deployed to the <drive>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\SiteTemplates\AXSITEDEF folder. The web part page templates are deployed to the language-specific site definition folder: <*drive*>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\<*lcid*>\AXSITEDEF.

Enterprise Portal deployment is deployed as a set of four SharePoint features. A SharePoint site represents a modular, server-side, file-system-level customization that contains items that can be installed and activated in a SharePoint environment. The feature definitions are deployed to <*drive*>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\FEATURES. These Enterprise Portal feature definitions are as follows:

- ***DynamicsAxEnterprisePortal*** Enables basic Enterprise Portal deployment steps, such as deploying the master page and other files and components, setting navigation providers, and registering Microsoft Dynamics AX. This feature is for the SharePoint Foundation environment.
- ***DynamicsAxEnterprisePortalMOSS*** Includes environment-specific steps for deploying to SharePoint Standard and Enterprise edition environments.
- ***DynamicsSearch*** Enables the Enterprise Portal search control on Enterprise Portal sites that enable searching across Microsoft Dynamics AX and SharePoint data.
- ***DynamicsAxWebParts*** Enables deployment of the various Microsoft Dynamics AX web parts.

Enterprise Portal feature-related files are stored in the AOT under Web\Web Files\Static Files. The *Static Files* node also has other infrastructure-related files, such as the *.aspx* file that is used for importing and exporting page and list definitions, document handling infrastructure files, the master page, common ASP.NET pages, images, style sheets, and configuration files.

EPSetupParams is an XML file used to define the default Enterprise Portal site attributes, such as title, description, and URL, when the site is automatically created through Enterprise Portal setup.

The Enterprise Portal master page automatically adds the Page title, Quick launch, and Infolog web parts. When a page is created in Enterprise Portal, these web parts are already available on the

webpage, creating consistency across all web part pages in Enterprise Portal and supporting rapid application development. Figure 7-15 shows some of the key files that constitute the site definition and their locations on the web server.

Enterprise Portal web parts are kept in the AOT under Web\Web Files\Static Files. If necessary, partners and customers can add their own web parts under this node, and Enterprise Portal will deploy these files to the global assembly cache on the web server and add a safe control entry in the Web.config file.

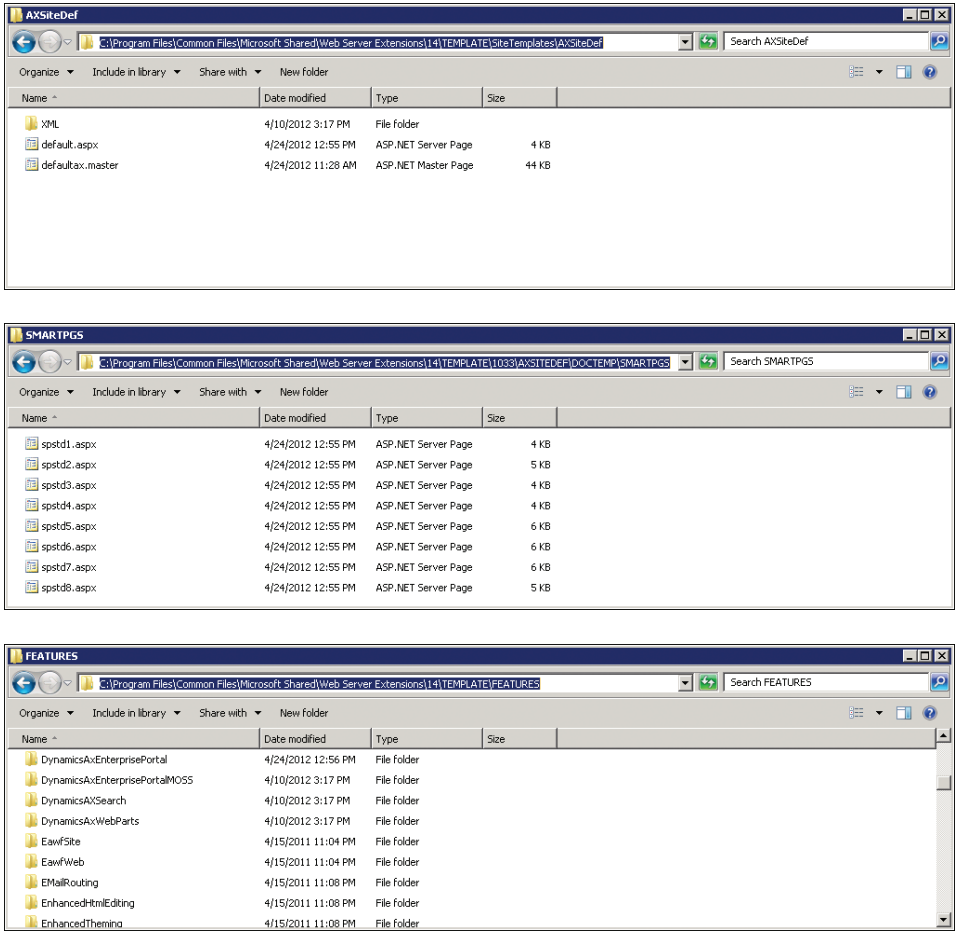


FIGURE 7-15 Enterprise Portal site definition files.

Web part pages display one or more web parts. Web parts provide an easy way to build powerful webpages that display a variety of information, ranging from a Microsoft Dynamics AX data view of a list in the current site to external data presented in custom-built web parts. You can create web part pages in SharePoint by using Windows Internet Explorer. You simply drag web parts onto web part pages and set their properties with prepopulated lists. You can edit web part pages in either SharePoint Designer or Internet Explorer. You can use Internet Explorer to edit a page and change

its web parts, arrange the order of the web parts, and set the web part properties. You can use SharePoint Designer to insert logos or other graphics, to customize document libraries or lists, to apply themes and styles, to customize the master page, and so on. Keep in mind, however, that you can't import pages edited with SharePoint Designer into the AOT.

You can import web part pages created in the Enterprise Portal site in SharePoint into the AOT as page definitions by using the Import Page tool from web menu items of type URL. The page definitions are stored in the AOT under Web\Web Files\Page Definitions.

The page definitions imported into the AOT automatically create pages when a site is created with the Enterprise Portal site definition. The *PublicPage* property of the page definition node determines whether the page should be created on the public site. All the pages are created for the authenticated site. The page definition *Title* property, if used, must be set to a label so that the page displays the localized title when used with different language settings.

Import and deploy a web part page

When you create a new web part page in Enterprise Portal, you should import the page into the AOT. You can then deploy the page to a different Enterprise Portal site or have the system automatically deploy it when creating a new Enterprise Portal site. To import the page to the AOT, create a web menu item that points to the page, right-click the item, and then click Import Page. The imported page definition is stored under \Web\Web Files\Page Definitions. Once imported, the pages can use Microsoft Dynamics AX labels for page titles so that the same page definitions can be used for sites in different languages.

To create or deploy an Enterprise Portal site or individual elements such as web modules, pages, web controls, images, and so on, you can use the AxUpdatePortal command-line utility. AxUpdatePortal also supports remote deployment, so you don't have to log on physically to an Enterprise Portal server to deploy to it.

The AxUpdatePortal utility is located either in the C:\Program Files\Microsoft Dynamics AX\60\ Setup folder where Enterprise Portal is installed or in the C:\Program Files\Microsoft Dynamics AX\60\ EnterprisePortalTools where the Microsoft Dynamics AX client is installed.

Table 7-3 lists the parameters that AxUpdatePortal supports.

TABLE 7-3 AxUpdatePortal utility parameters.

Parameter	Description
Listvirtualservers	Lists all virtual servers (SharePoint web applications and IIS websites) on the server
-deploy	Deploys a new virtual server (SharePoint web application) to an IIS web server that already has Enterprise Portal installed
-createsite	Creates an Enterprise Portal website within an existing Enterprise Portal virtual server
-updateall	Updates all web components on an Enterprise Portal website (SharePoint site collection)

Parameter	Description
-proxies	Update all proxies on an Enterprise Portal website (SharePoint site collection)
-images	Updates all images on an Enterprise Portal website (SharePoint site collection)
-updatewebcomponent	Updates a web component on an Enterprise Portal website (SharePoint site collection)
-websiteurl <value>	The URL of an Enterprise Portal website (SharePoint site collection)
-treenodepath <value>	The tree node path of the web component to deploy
-updatewebsites	Updates all Enterprise Portal websites during a redeployment
-iisreset	Stops and restarts the IIS web server after completing the deploy operation

For more details about these parameters, use *AxUpdatePortal /?*.

Here are some examples of how to use the AxUpdatePortal utility to perform actions on a website located at *http://ServerName/site/DynamicsAX*.

Create and deploy a new Enterprise Portal website:

```
AxUpdatePortal -deploy -createsite -websiteurl "http://ServerName/site/DynamicsAx"
```

Update all components of the Enterprise Portal website:

```
AxUpdatePortal -updateall -websiteurl "http://ServerName/site/DynamicsAx"
```

Deploy all proxies to the Enterprise Portal website:

```
AxUpdatePortal -proxies -websiteurl "http://ServerName/site/DynamicsAx"
```

Deploy the Customers web control to the Enterprise Portal website:

```
AxUpdatePortal -updatewebcomponent -treenodepath "\Web\Web Files\Web Controls\Customers"
-websiteurl "http://ServerName/site/DynamicsAx"
```

Enterprise Search

Enterprise Search in Microsoft Dynamics AX 2012 lets users search for data, metadata, and the contents of documents that are attached to records. This search capability is available in both Enterprise Portal and the Microsoft Dynamics AX client. Enterprise Search uses the Metadata service and the Query service in Microsoft Dynamics AX to gather the data and metadata from Microsoft Dynamics AX. To index and execute search queries, Enterprise Search uses the SharePoint Business Connectivity Services (BCS).

To enable this rich search functionality, you must install the Enterprise Search component in Microsoft Dynamics AX Setup. If you are using SharePoint Standard or Enterprise editions, you do not need to install any other prerequisites for Enterprise Search because these have search capabilities built-in. However, if you are using SharePoint Foundation, you need to install Microsoft Search Server Express as a prerequisite for Enterprise Search.

Enterprise Search uses queries to make data searchable in Microsoft Dynamics AX. When Enterprise Search is installed, it indexes the default queries and runs a full crawl of the data and metadata. If you want to make additional data searchable, use the following sequence of steps:

1. In the AOT, either find the query that fetches the data, or create a new query.
2. Set the *Searchable* property on the query to *Yes*.
3. Compile the query and ensure that there are no best practice errors.
4. In the Microsoft Dynamics AX client, start the Enterprise Search Configuration Wizard (System Administration > Setup > Search > Search Configuration), which is shown in Figure 7-16. The wizard will display a list of all queries in the AOT whose *Searchable* property is set to *Yes*.

   By default, all queries whose *Searchable* property is set to *Yes* are selected to be published to Microsoft Business Connectivity Services (BCS). You can clear any queries that you do not want to publish.
5. If you wish, use the Select Fields option to prevent specific fields from being indexed.
6. Select the check box to start a full crawl of the data source. Alternatively, you can use SharePoint Central Administration to start a full or incremental crawl manually.
7. Click Next, and then click Finish.

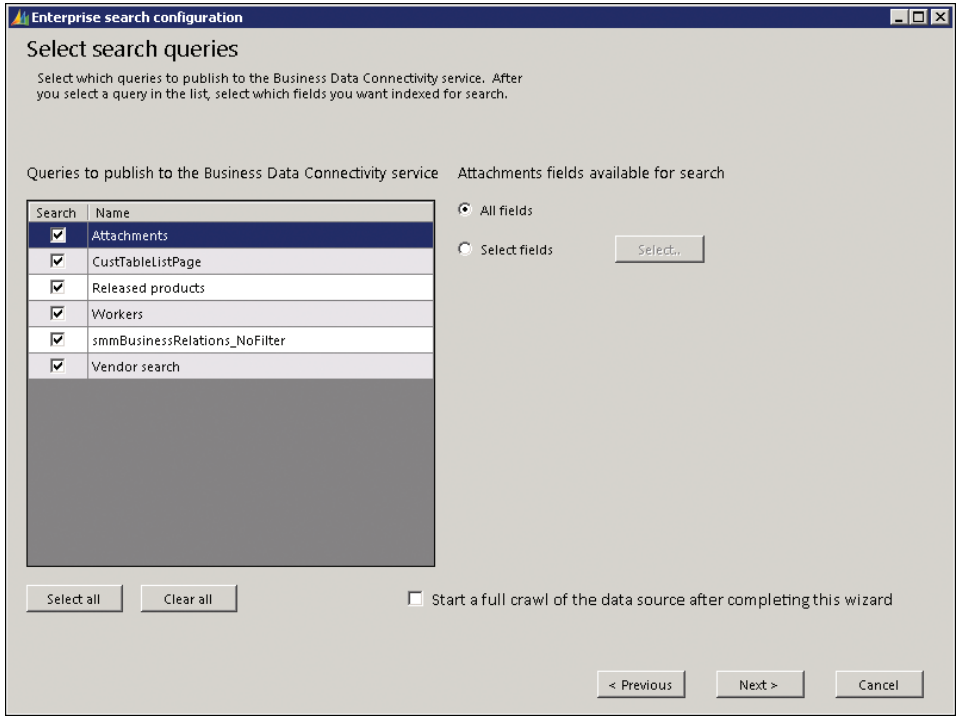


FIGURE 7-16 Enterprise Search Configuration Wizard.

The Enterprise Search Configuration Wizard uses the credentials of a search crawler account to index the data and publishes the queries to BCS. You can also see your published queries in SharePoint Central Administration under Application Management > Manage Service Applications > Business Data Connectivity Service.

Changes to metadata information (such as web menus and so on) are rare, so by default, Enterprise Search executes a full crawl of the metadata only once during installation. On the other hand, changes to data (such as sales orders and so on), are frequent. By default, Enterprise Search performs a full crawl of the data once during installation, and an incremental crawl every day at midnight.

If you publish a new query, you can start a full crawl directly from the Enterprise Search Configuration Wizard, as mentioned earlier. You can also start a full or an incremental crawl manually in SharePoint Central Administration by following these steps:

1. Start SharePoint Central Administration.
2. Navigate to Application Management > Manage Service Applications > Search Service Application.
3. In the left navigation pane, under Crawling, click Content Sources. You will see two content sources: one for data and one for metadata.
4. Click on either content source, and then click either Start Full Crawl or Start Incremental Crawl, as shown in Figure 7-17. Keep in mind that crawling can take a long time, depending on how much data or metadata there is to index.

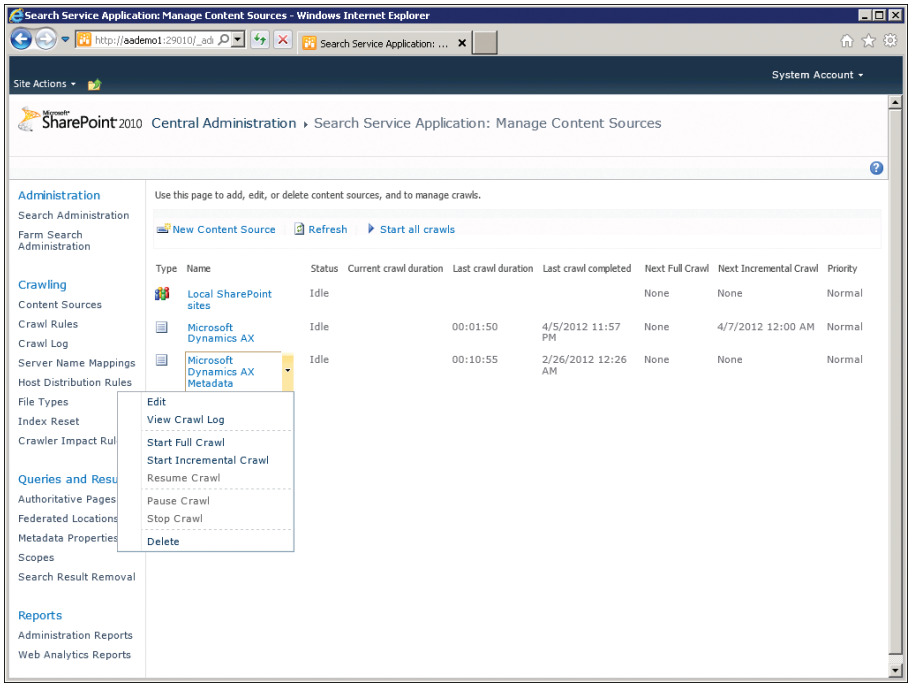


FIGURE 7-17 Starting a crawl by using SharePoint Central Administration.

After the data and metadata has been crawled and indexed, it is published to BCS. Users can execute searches by using the search box located in the upper-right corner of Enterprise Portal. The results are trimmed at search time based on the user's role, language, and other settings, so that users see only the data that is available and applicable to them.

Themes

Enterprise Portal integrates with SharePoint themes. You can apply an existing SharePoint theme to the Enterprise Portal site to change its appearance just like any other SharePoint site. Partners and customers can also create new SharePoint themes and customize or extend the Enterprise Portal style sheets to map to the new theme.

Enterprise Portal uses five style sheets. AXEP.css is the base style sheet. AXEP_RTL.css is used for right-to-left languages and cascades on top of AXEP.css. These two files are located on the web server under <drive>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\<lcid>\STYLES\Themable. The AXEP_CRC.css, AXEP_CRC_RTL.css, and AXEP_WebPart_Padding.css style sheets are used for Role Centers when rendered on the Microsoft Dynamics AX client. These files are located on the web server under *<drive>*:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ep\Stylesheets.

The Enterprise Portal master page references these style sheets. The AXEP.css and AXEP_RTL.css style sheets contain SharePoint theme directives and are therefore placed in the special directory, where SharePoint can locate them.

When a SharePoint theme is applied to the Enterprise Portal site, SharePoint parses the directives and makes modifications to reflect the new theme. These modifications include color and font replacements and even recoloring of some images. It then stores the modified style sheet and images in the SharePoint content database. The master page then references this new style sheet so Enterprise Portal appearance reflects the applied theme.

CHAPTER 8

Workflow in Microsoft Dynamics AX

In this chapter

Introduction 245
Microsoft Dynamics AX 2012 workflow infrastructure 246
Windows Workflow Foundation 249
Key workflow concepts 250
Workflow architecture 256
Workflow life cycle 262
Implementing workflows 263

Introduction

Few people would deny the importance or significance of the processes that drive the businesses and organizations that we work for and interact with on a daily basis. *Business processes* represent the key activities that, when carried out, are intended to achieve a specific goal of value for the business or organization. For example:

- A manufacturing operation in which business process activities include design, development, quality assurance testing, and delivery of a saleable (and hopefully profitable) range of goods
- Sales process activities for manufactured items, including marketing, locating prospects, providing quotes, converting quotes to orders and prospects to customers, shipping the product, invoicing, and obtaining payment
- Supporting activities that contribute to the business or organization in tangible ways, such as hiring new employees and managing employee expenses

Viewing activities in terms of the business processes that encompass them affords businesses and organizations the opportunity to systematically define, design, execute, evaluate, and then improve the way that these activities are performed. This systematic approach is extremely valuable, even critical, given that today's businesses and organizations have to react to an increasingly rapid rate of change and the ever-expanding influence of globalization.

Enterprise resource planning (ERP) suites, such as Microsoft Dynamics AX, exist to automate business processes and to provide the capability to adapt these processes to the needs of businesses and organizations over time. Before Microsoft Dynamics AX 2009, no standard workflow infrastructure existed in the product, and each company had to write specific business logic to implement everyday activities

such as approvals. The Microsoft Dynamics AX 2009 release included a built-in workflow infrastructure to make it easier for businesses and organizations to automate and manage business processes. This infrastructure has been enhanced further in Microsoft Dynamics AX 2012.

The main difference between *business processes* and *workflows* (these terms are often used interchangeably) is their scope, level of abstraction, and purpose. Business processes represent the broad set of activities that a business or organization needs to carry out, along with the interrelationships among the activities. Business processes are implementation-independent and can combine manual and automated activities. Workflows represent the automated parts of a business process that coordinate various human or system (or both) activities to achieve a particular outcome, and they are implementation-specific. Therefore, workflows are used to implement parts of a business process.

Microsoft Dynamics AX 2012 workflow infrastructure

Fundamentally, a workflow consists of one or more activities that represent the items of work to be completed. In addition, the concept of workflows that connect the activities and govern the sequence of execution (referred to as the *structure* of a workflow) is key. The behavior of a workflow is determined by its type. Figure 8-1 illustrates the major types of workflows and identifies where the emphasis of the workflow infrastructure is located in Microsoft Dynamics AX 2012.

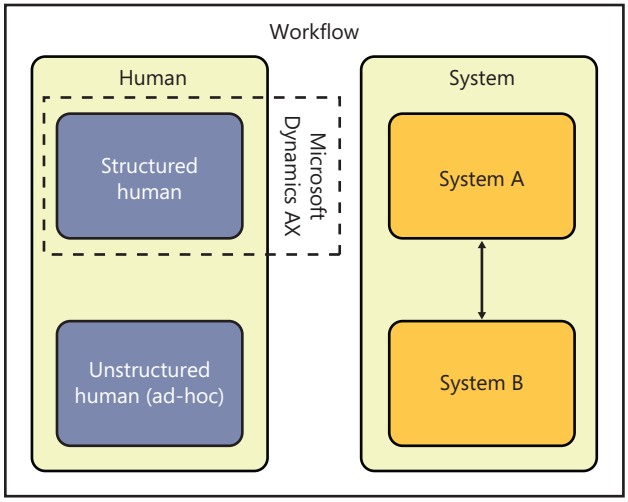


FIGURE 8-1 Major types of workflows.

A major distinction exists between *human workflows* and *system workflows*. (For more information, see the "Types of workflows" sidebar on the next page.) Workflows in Microsoft Dynamics AX 2012 are primarily designed to support structured human workflows. Almost 60 structured human workflows are included with the product, spanning accounts payable, accounts receivable, budgeting, fixed assets, general ledger, organization administration, procurement and sourcing, system administration, time and attendance, and travel and expense. Whereas the built-in workflows tend

to focus on structured human workflows that obtain *approvals*, you can also create workflows that contain tasks for humans to complete or a mixture of structured and unstructured tasks along with approvals and automated tasks. Customers, partners, and independent software vendors (ISVs) can create additional workflows to supplement those in the product.

Types of workflows

Workflows are divided into two major types: human and system. This sidebar examines some of the basic differences between the two types.

Human workflows

A key attribute of a human workflow is that people are involved in the workflow as it executes; in other words, a human workflow is generally interactive, although it might contain activities that are non-interactive, such as automated tasks. Most often, the interaction takes the form of responding to a workflow notification and taking an action of some kind, such as approving or rejecting the business document being processed. Human workflows can be subdivided further into structured and unstructured types. Structured human workflows are used for processes in which execution must be repeatable and consistent over time, such as expense approval and purchase requisition processing. Structure is important because to improve a business process, you must have a way to measure the performance of the workflows that are executed to automate that business process. If a workflow isn't structured for repeatability and consistency, you are going to have a difficult time identifying what to improve.

Unstructured human workflows differ from structured ones in that the exact activity flow doesn't have to be defined up front—but it should be possible to easily establish and assign to the required people. An example of an unstructured human workflow is reviewing a document where the participants and the type of approval required are decided just before the workflow starts. This type of human workflow is less useful to analyze for process improvement because each unstructured workflow might operate differently, depending on how it is used. However, an unstructured human workflow does help coordinate human activities.

System workflows

A system workflow is a non-interactive workflow that automates a process that spans multiple systems, such as transferring an order from one system to another. Generally, such workflows are structured because they must be consistently repeatable.

You often need to combine human and system workflows to implement a given business process. For example, expense reports must be approved, and then the expense lines must be posted after approval.

Because existing Microsoft Dynamics AX modules use approvals extensively, the workflow infrastructure in Microsoft Dynamics AX 2012 is primarily intended to support structured human workflows. Focusing on this type of workflow lays the groundwork to help businesses and organizations more easily automate, analyze, and improve high-volume workflows across their ERP systems.

Each structured human workflow in Microsoft Dynamics AX 2012 acts on a single document type because data is the key constituent of an ERP system. (Think of the broad categories of data that exist in an ERP system: master data, transaction data, and reference data. Processes that operate within those systems are largely data-driven.)

Here are some of the key tasks that you can perform with structured human workflows in Microsoft Dynamics AX 2012:

- Define the activities that must take place, based on the business process that is being automated.
- Define the sequence in which tasks, approvals, subworkflows, and the new workflow elements in Microsoft Dynamics AX 2012 (manual decisions, automated decisions, parallel activities and branches, automated tasks, and line-item workflows) execute to reflect the order in which activities must be completed in a business or an organization.
- Set up a condition to determine which workflow to use in a given situation.
- Decide how to assign an activity to users.
- Specify the text that is displayed in the user interface for the various activities to help users understand what they need to do.
- Define a set of outcomes for an activity that users can select from.
- Select which notifications to send, which email template to use, when to send the notifications, and who should receive the notifications.
- Establish how a workflow should be escalated if there is no timely response to an activity.

Four types of users interact with the workflow infrastructure in Microsoft Dynamics AX 2012: business process owners, developers, system administrators, and end users (called "*users*" in this book).

Business process owners and developers are primarily responsible for defining, designing, and developing workflows, whereas system administrators and users interact with workflows that are executing.

- **Business process owners** understand the objectives of the business or organization within which they operate to the degree that they can envision how best to structure the activities within their areas of responsibility. Business process owners therefore configure workflows that have already been implemented and work with functional consultants or developers to enable other modules or create new workflow types in existing modules.
- **Developers** work with business process owners to design and implement any underlying business logic that is required to support workflows that are being developed.
- **System administrators** set up and maintain the development and production environments, ensure that the workflow infrastructure is configured correctly, monitor workflows as they execute, and take actions to resolve issues with workflows that are executing.

- **Users** interact with workflows when necessary; for example, by submitting a business document record, taking a particular action (such as approving or rejecting a document), entering comments, viewing workflow history, and so on.

Windows Workflow Foundation

The workflow infrastructure in Microsoft Dynamics AX 2012 is related to Windows Workflow Foundation (WF), which is part of the Microsoft .NET Framework 4. WF provides many fundamental capabilities that are used by the workflow infrastructure in Microsoft Dynamics AX 2012. As a low-level infrastructure component, however, WF has no direct awareness of or integration with Microsoft Dynamics AX 2012. In Figure 8-2, the workflow infrastructure (A) is an abstraction layer that sits above WF (B) and allows workflows that are specific to Microsoft Dynamics AX to be designed, implemented, and configured in Microsoft Dynamics AX 2012 and then executed by using WF.

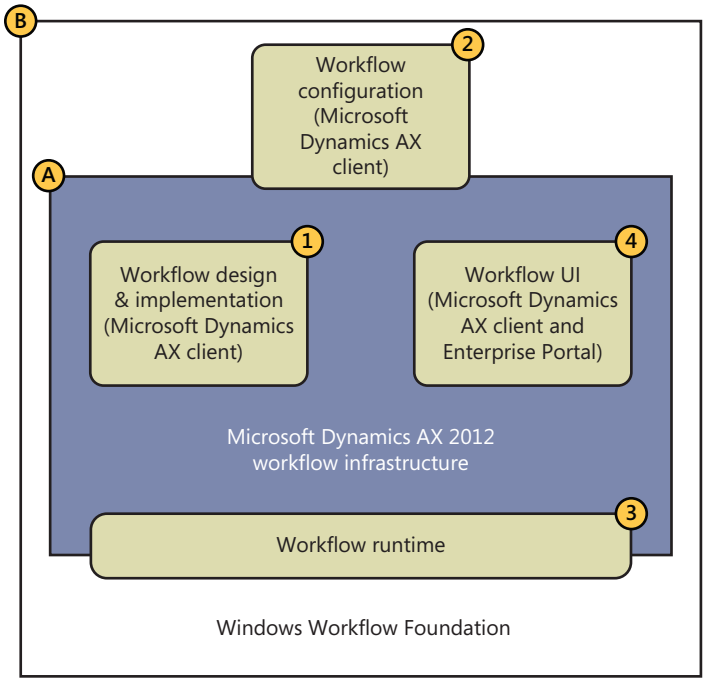


FIGURE 8-2 Relationship between the Microsoft Dynamics AX 2012 workflow infrastructure and WF.

In the following list, each numbered item refers to the corresponding part of Figure 8-2:

1. The developer designs and implements workflow elements and business logic in the Application Object Tree (AOT).
2. The business process owner models workflows using the new graphical workflow editor in the Microsoft Dynamics AX 2012 client, which is based on the WF Designer.

3. The workflow runtime bridges both the Microsoft Dynamics AX 2012 workflow infrastructure and WF; it instantiates and then executes workflows. (The system administrator manages the runtime environments.)

4. Users interact with workflow user interface (UI) controls both in the Microsoft Dynamics AX 2012 Windows client and in the Enterprise Portal web client.

Key workflow concepts

As a Microsoft Dynamics AX developer, workflow is something that you work with to help the users in a business or organization improve their efficiency. The ultimate goal for workflow in Microsoft Dynamics AX 2012 is to make it as easy as possible for business process owners to configure workflows fully themselves, freeing developers to work on other activities. Currently, developers and business process owners work together to create and customize workflows.

You need to understand a number of key concepts to help business process owners implement workflows successfully.

Workflow document and workflow document class

The workflow document, sometimes referred to as the *business document*, is the focal point for workflows in Microsoft Dynamics AX 2012. Every workflow type and every workflow element must reference a workflow document because it provides the data context for the workflow. A workflow document is an AOT query supplemented by a class in the AOT (referred to as the *workflow document class*). The term *workflow document* is used instead of *query* because it more accurately portrays what the workflow operates on. The query used by a workflow document can reference multiple data sources and isn't constrained to a single table. In fact, a query can reference data sources hierarchically. However, if there are multiple data sources within a query, the first data source is considered the *primary* or *root* data source.

Tip The workflow document and workflow document class are located in the AOT in the Microsoft Dynamics AX 2012 client.

Workflow in Microsoft Dynamics AX 2012 incorporates an expression builder that you can use to define conditions that control the behavior of an executing workflow. The expression builder uses the workflow document to enumerate the fields that can be referenced in conditions. To make derived data available within conditions, you add *parm* methods to the workflow document class, and then add X++ code to the *parm* methods to produce the derived data. The workflow document then returns the fields from the underlying query plus the data generated by the *parm* methods.

Workflow categories

Workflow categories determine the association a workflow type has to a specific module. (Without these categories, you would see all workflows in the context of every module in Microsoft Dynamics AX 2012.) For example, a workflow category named *ExpenseManagement*, which is mapped to the Travel and Expense module, comes with Microsoft Dynamics AX 2012. All workflows associated with this module are visible in the Microsoft Dynamics AX 2012 client within the Travel and Expense module. If you add a new module to Microsoft Dynamics AX 2012, you must create a new module and a new workflow category that references that module.

Tip Workflow categories are located in the AOT in the Microsoft Dynamics AX 2012 client.

Workflow types

The workflow type (called a "*template*" in Microsoft Dynamics AX 2009) is the primary building block that developers use to create workflows. You generate the workflow type by using the new Workflow wizard, shown in Figure 8-3. The wizard automates the creation of the metadata required for a workflow type; all you need to do is specify the name, workflow category, query, and menu items.

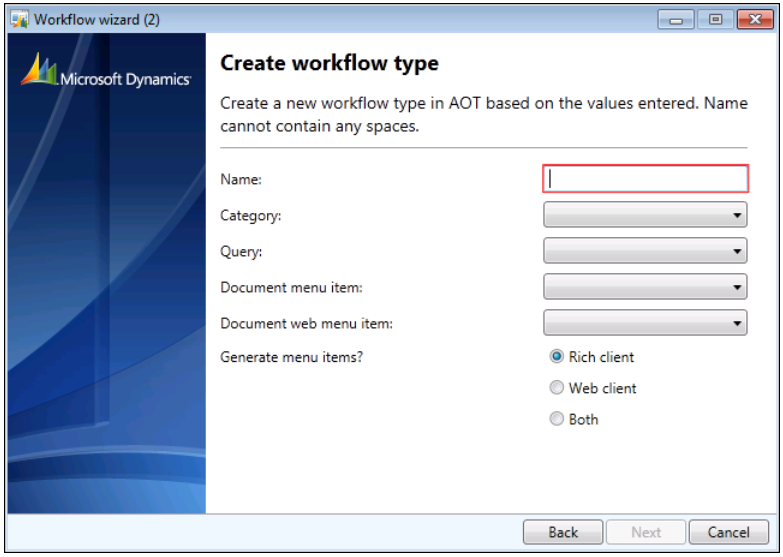


FIGURE 8-3 The Create Workflow Type page in the Workflow wizard.

The resulting metadata is created under the AOT\Workflow\Workflow Types node. The business process owner later references this workflow type when creating an actual workflow.

Tip Workflow types are located in the AOT in the Microsoft Dynamics AX 2012 client.

For more information about the Workflow wizard, see "How to: Create a New Workflow Type" at *http://msdn.microsoft.com/en-us/library/cc594095.aspx*.

Event handlers

Event handlers are well-defined integration points that developers use to trigger application-specific business logic during workflow execution. Workflow events are exposed at the workflow level and the workflow element level. For more information about event handlers, including where they are used, see "Workflow Events Overview" at *http://msdn.microsoft.com/en-us/library/cc588240.aspx*.

Tip Event handlers are located in the AOT in the Microsoft Dynamics AX 2012 client.

Menu items

Workflow in Microsoft Dynamics AX 2012 uses both display and action menu items. Display menu items are used to navigate to a form in the Microsoft Dynamics AX 2012 client that displays the details of the record being processed by workflow. Web menu items are used to navigate to the same type of webpage in Enterprise Portal. Action menu items are used for each possible action that a user can take in relation to a workflow. They also provide another integration point for you to integrate custom code. For more information about the menu items that are used in the workflow infrastructure, see "How to: Associate an Action Menu Item with a Workflow Task or Approval Outcome" (*http://msdn.microsoft.com/en-us/library/cc602158.aspx)* and "How to: Associate a Display Menu item with a Workflow Task or Approval" *(http://msdn.microsoft.com/en-us/library/cc604521.aspx)*.

Tip Menu items are located in the AOT in the Microsoft Dynamics AX 2012 client.

Workflow elements

The elements of a workflow represent the activities within the workflow. The business process owner models these elements. An element can be a task, an approval, a subworkflow, a manual decision, an automated decision, a parallel activity with multiple branches, a line-item workflow, or an automated task. Developers implement the task, approval, line-item workflow, and automated task elements. The rest are referred to as "configuration only" elements that business process owners can use in the graphical workflow editor. The following list describes each element:

- **Tasks** are generic workflow elements that represent a single unit of work. The developer defines the possible outcomes for each task.
- **Approvals** are specialized tasks that allow sequencing of multiple steps and use a fixed set of outcomes.

- **Subworkflows** are workflows that are invoked from other workflows.
- **Manual decisions** enable the workflow to follow one of two possible paths based on an action taken by a user.
- **Automated decisions** enable the workflow to follow one of two possible paths based on a condition.
- **Parallel activities** contain two or more branches that represent discrete workflows, and they are executed simultaneously.
- **Line-item workflows** are modeled within a workflow that exists for a business document that represents the master in a master-detail relationship. They enable specific workflows to be instantiated on line items that are associated with the master business document; for example, expense lines on an expense report.
- **Automated tasks** are non-interactive and invoke X++ business logic synchronously.

Manual and automated decisions, parallel activity, line-item workflows, and automated tasks are new in Microsoft Dynamics AX 2012. In addition workflow wizards have been added to make the creation of approval and task elements easier.

Tip Workflow elements are located in the AOT in the Microsoft Dynamics AX 2012 client.

Queues

The ability to assign workflow work items to a queue is new in Microsoft Dynamics AX 2012. Queues offer an alternative to assigning workflow work items directly to users by providing support for teams that collaborative within a business process. With this approach, a work item is first assigned to a queue; then, the work item is claimed by a member of the queue so it can be worked on. The eventual work item owner can also return the work item to the original queue, put the work item in another queue, or assign the work item to another user.

To use work item queues, complete the following steps:

1. Create one or more work item queues for a selected workflow document (for example, purchase requisition header) and assign one or more Microsoft Dynamics AX users to each queue. These assigned users can view and take action on work items assigned to the queue. Each queue also has an administrator, which by default is the user who created the queue.
2. Create a work item group, a container for grouping one or more work item queues, and then add all of the work item queues for a given document type to the work item group.
3. Set the status of the work item queues to Active so that workflows can assign work items to them.

4. Create a workflow by using a workflow type based on the same business document as the queue. Model a task element within the workflow and configure it to be assigned to the appropriate work item queue.

Note Only work items generated from task elements can be assigned to queues, because a task can be completed by only a single user. In this case, the queue provides a way for users to assign themselves to work items. Approvals differ from tasks in this respect because approvals are explicitly modeled around an approval pattern that consists of one or more discrete steps; each step has a specific type of user assignment and a completion policy that controls when an approval is complete.

5. Submit a record to workflow. Any work items created for the task element are directed to the appropriate queue. Users can access work item forms in the Microsoft Dynamics AX 2012 client and review, accept, and take action on work items in their queue.

For more information about setting up work item queues, see "Configure work item queues" at *http://msdn.microsoft.com/en-us/library/gg731875.aspx*.

Providers

Workflow in Microsoft Dynamics AX 2012 uses the provider model as a flexible way of allowing application-specific code to be invoked for different purposes when a workflow is executing. There are four provider types within the workflow infrastructure: due date, participant, hierarchy, and queue. The way in which provider metadata is stored has changed in this release. In Microsoft Dynamics AX 2009, providers were developed as classes that implemented a provider interface and were registered on the workflow element as a property. Workflow providers in Microsoft Dynamics 2012 now have their own node in the AOT (AOT > Workflow > Providers), as shown in Figure 8-4.

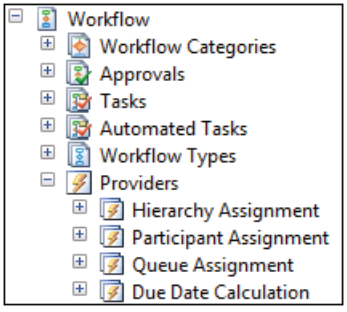


FIGURE 8-4 The new Workflow Providers node in the AOT.

In addition each provider now has properties that are used to define the following:

- Their organization scope (AssociationType)
- Whether the provider applies to all workflow types or specific types (Workflow Types subnode)

For more information about workflow providers, including where they are used, see "Workflow Providers Overview" at *http://msdn.microsoft.com/en-us/library/cc519521.aspx*.

Workflows

The business process owner creates workflows using the new graphical workflow editor (shown in Figure 8-5) in the Microsoft Dynamics AX 2012 client. The business process owner first selects a workflow type and then configures the approvals, tasks, and other elements that control the flow of activities through the workflow.

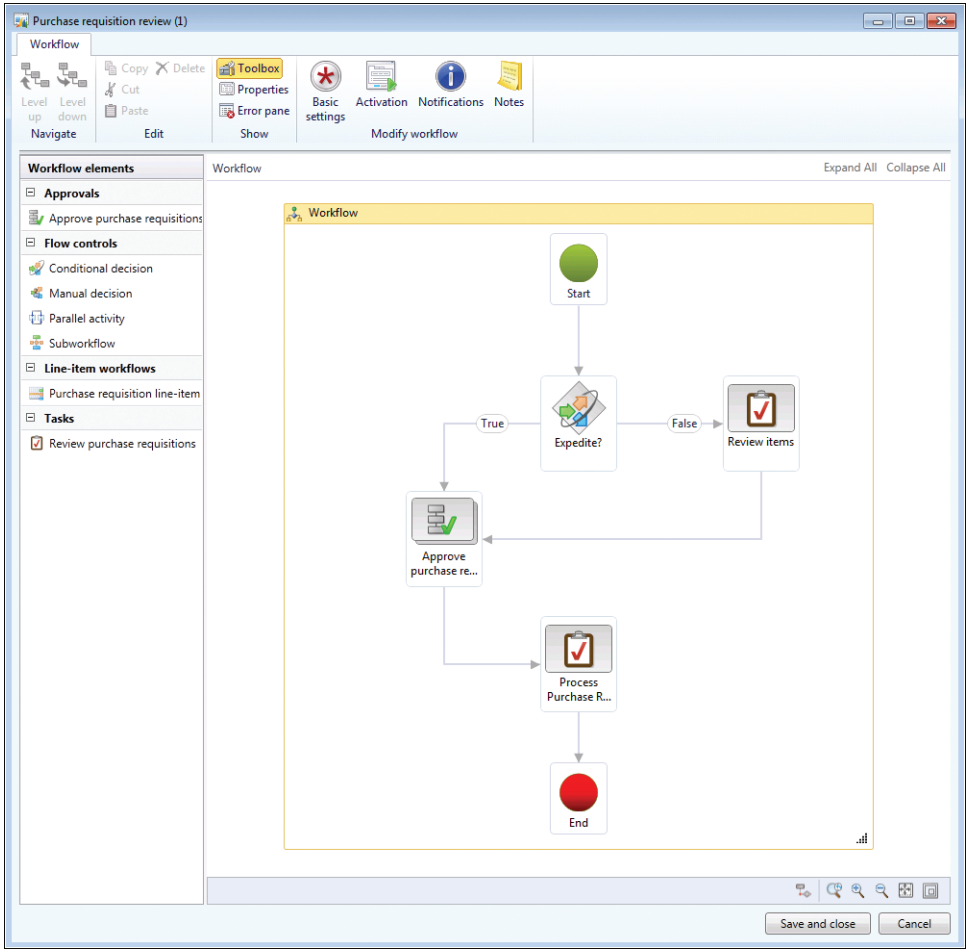


FIGURE 8-5 The new graphical workflow editor.

Workflows are located in the Microsoft Dynamics AX 2012 client. A list page containing the workflows for a given module is located on the area page for the module under Setup > [*module name*] workflows.

Workflow instances

A workflow instance is an activated workflow created by combining the workflow and the underlying AOT workflow elements on which the workflow is based (the workflow type, tasks, and approvals).

Workflow instances are located in the Microsoft Dynamics AX 2012 workflow runtime.

Work items

Work items are the actionable units of work that are created by the workflow instance at run time. When a user interacts with a workflow, he or she responds to a work item that has been generated from a task element, an approval element, or a manual decision. Work items are displayed in the *Unified worklist* web part and in the Microsoft Dynamics AX 2012 client.

Workflow architecture

Microsoft designed the workflow infrastructure based on a set of assumptions and goals related to the functionality it wanted to deliver. Two assumptions are the most significant:

- Business logic (X++ code) invoked by workflow is always executed on the Application Object Server (AOS).
- Workflow orchestration is managed by WF in the .NET Framework 4.0.

The first assumption reflects the fact that most business logic already resides and is executed on the AOS. The second assumption is based on the opportunity to use existing Microsoft technology for orchestrating workflows in Microsoft Dynamics AX 2012 instead of designing and implementing this functionality from scratch. In Microsoft Dynamics AX 2012, the WF framework was integrated into the AOS.

The following primary goals influenced the architecture:

- Create an extensible, pluggable model for workflow integration (including events and providers), because the workflow *infrastructure* had to be flexible enough to address application-specific requirements as they pertain to workflow execution.
- Build in scalability that accommodates the growth of workflow usage in Microsoft Dynamics AX 2012 over time and provides options for scale up and scale out.
- Minimize the performance impact on transactional X++ business logic to invoke workflows. For example, if workflow activation is triggered from saving a document, no adverse

performance side effects should result from doing this in the same physical transaction (*ttsbegin/ttscommit*) as the save operation.

The next section expands on the capabilities of the workflow runtime in Microsoft Dynamics AX 2012.

Workflow runtime

Figure 8-6 shows the components of the workflow runtime and their interaction.

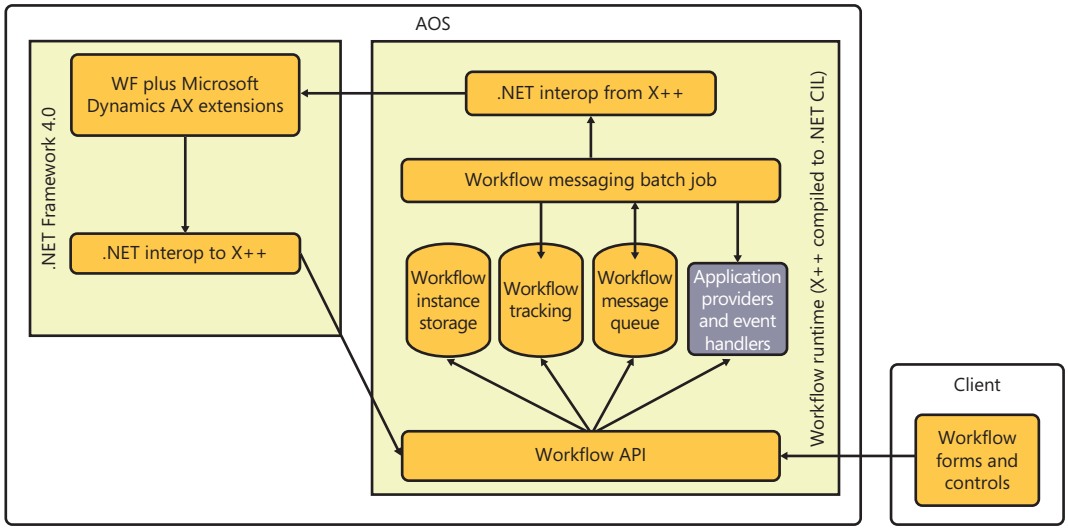


FIGURE 8-6 Workflow runtime.

The workflow runtime includes the following components:

- **Workflow API** An application programming interface (API) that exposes the underlying workflow functionality to the rest of Microsoft Dynamics AX 2012.
- **Workflow instance storage** Tables that store the serialized workflow instance data. Whenever the workflow goes idle waiting for a user or a system action, the workflow instance is serialized and saved to the database and removed from memory on the AOS.
- **Workflow tracking** Tables that store the tracking information for a workflow instance. Tracking information is used to display historical information of completed and pending workflow instances.
- **Workflow message queue** A table that stores the messages used for communication between the .NET Framework 4.0 workflow instance and the Microsoft Dynamics AX 2012 workflow runtime in X++. A message exchange is required for any scenario where transactional X++ application logic must execute as part of a workflow instance or when user action is required.
- **Application providers and event handlers** Application code that is invoked by the workflow instance.

- **Workflow messaging batch job** A server-bound batch job that is dedicated to processing messages from the workflow message queue. This batch job supports parallel processing of batch tasks to enable both scaling up and scaling out for workflow processing. The batch job runs in X++ compiled into .NET common intermediate language (CIL).
- **WF plus Microsoft Dynamics AX extensions** The workflow framework provided by the .NET Framework 4.0 together with the Microsoft Dynamics AX custom workflow activities, custom providers, and custom workflow host.

Workflow runtime interaction

Figure 8-7 shows the logical control flow the workflow runtime uses to process a workflow activation message.

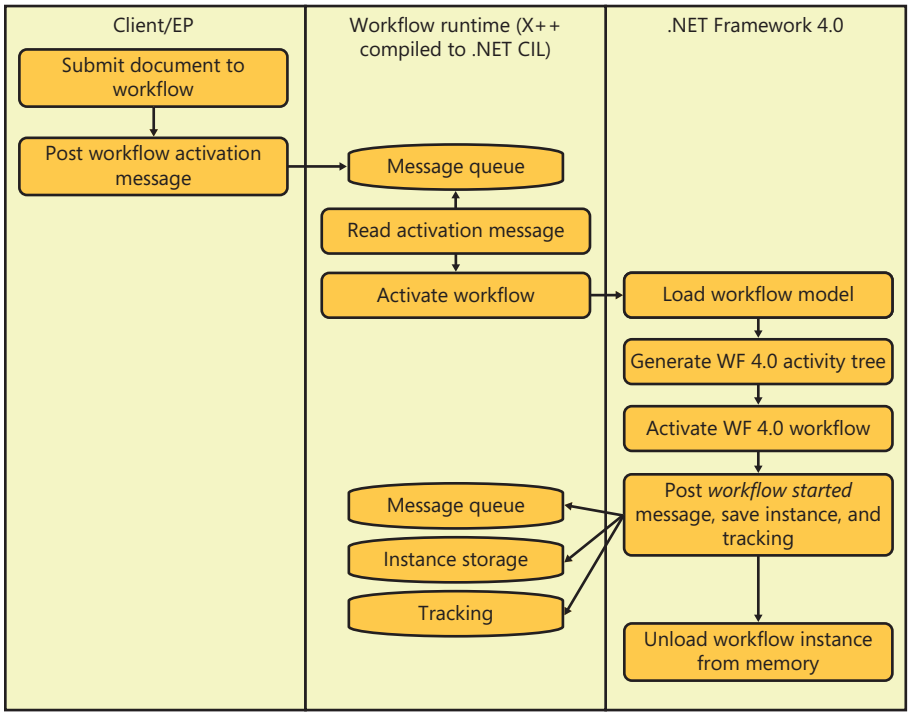


FIGURE 8-7 Logical workflow runtime control flow used to process a workflow activation message.

As an example of how these components interact at run time, the following sequence explains what happens when a user clicks Submit to activate a workflow on a record in Microsoft Dynamics AX 2012:

1. The Submit action invokes the Workflow API to post a workflow activation message for the selected workflow. This causes a message to be posted into the message queue.
2. The message is processed by the messaging batch job that calls the workflow runtime in the .NET Framework 4.0 to activate the workflow.

3. The Microsoft Dynamics AX extensions to the .NET Framework 4.0 receive the request and first load the workflow model. The workflow model is the structural representation of the workflow along with all of the workflow and workflow element properties. This model was created using the Microsoft Dynamics AX 2012 graphical workflow editor.

4. From the workflow model, the .NET Framework 4.0 workflow activity tree is built. These are the runtime workflow activities that orchestrate the workflow. These activities are a combination of custom Microsoft Dynamics AX 2012 activities and the primitive activities from .NET Framework 4.0.

5. Once the workflow reaches the first point where application logic may need to run, a message is posted, the workflow instance state is serialized and saved, and the workflow tracking data is updated. The *workflow started* message is posted at this point.

6. After the workflow instance goes idle and is saved to the database, it is removed from memory in the AOS to save physical computer resources. The workflow instance is brought back into memory after the *workflow started* message is processed and the *acknowledge workflow started* message is posted and begins to be processed.

Figure 8-8 shows the logical workflow runtime control flow to process a *workflow started* message.

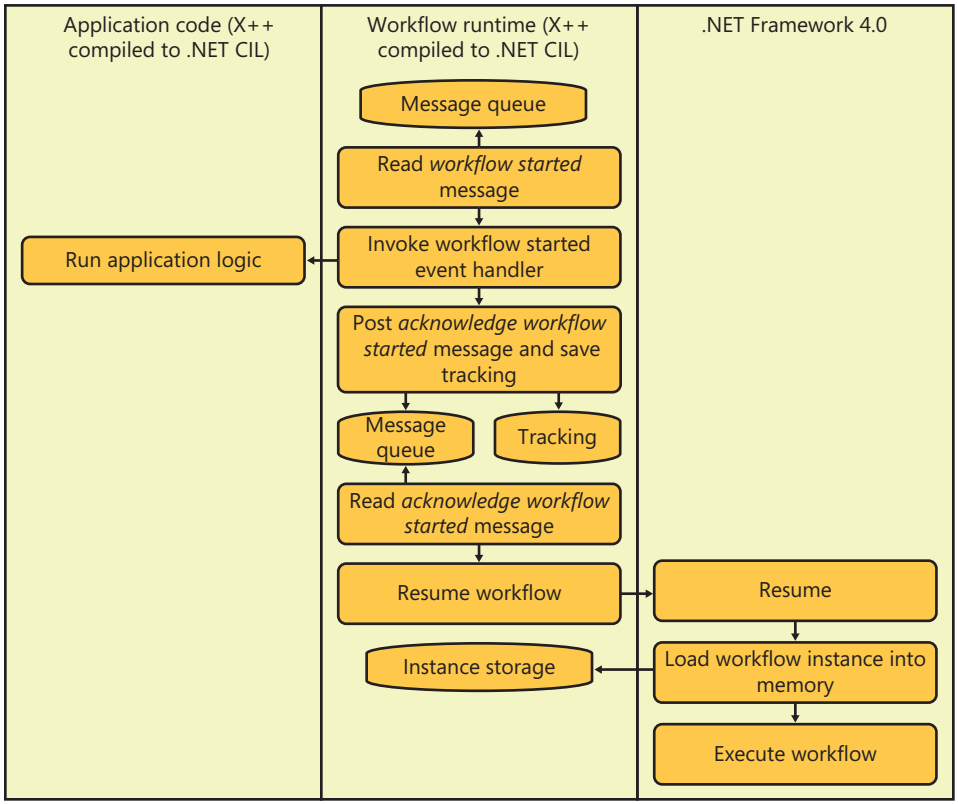


FIGURE 8-8 Processing a *workflow started* message.

The flow in Figure 8-8 builds on the flow in Figure 8-7. A *workflow started* message is currently posted to the message queue and is ready for processing as follows:

1. The workflow messaging batch job reads the *workflow started* message from the message queue. This message was posted by the workflow instance to allow application logic that was registered for this event to be invoked.
2. The application event handler that was registered for the *workflow started* event is invoked. This event handler runs the necessary X++ business logic to update the state of the underlying document.
3. An *acknowledge workflow started* message is posted and the *workflow started* message is removed from the message queue. Workflow tracking information is also logged at this point.
4. The workflow messaging batch job reads the *acknowledge workflow started* message and calls the workflow runtime in .NET Framework 4.0 to resume the workflow instance.
5. Microsoft Dynamics AX extensions to .NET Framework 4.0 receive the request to resume and then load the serialized workflow instance state from the workflow instance storage. This action brings the workflow instance back into memory on the AOS.
6. The workflow instance is placed back into the .NET Framework 4.0 workflow scheduler to be executed. The workflow then executes until the next point that X++ application logic needs to be invoked or until the workflow assigns work to users. Both of these represent points in the workflow instance where the instance must wait for either a system action (for example, processing the application event handler) or a human action (for example, a user approving an expense report).

Logical approval and task workflows

Another way to visualize how the key workflow concepts and architecture come together is to look at the interaction patterns of approval and task elements at run time. Four main types of interactions can occur: workflow events, acknowledgments (of events), provider callbacks, and infrastructure callbacks.

- **Workflow event** The workflow instance posts a message, saves the workflow instance, inserts tracking information, and removes the originating message. The workflow instance then waits for the corresponding message to be processed. The workflow messaging batch job processes the message by invoking the event handler on the corresponding workflow type, task, approval, or automated task. Then the workflow messaging batch job posts the acknowledgment message.
- **Acknowledgment** An acknowledgment message is the response to an event triggered from a workflow instance. Upon receiving the acknowledgment, the workflow instance is loaded from workflow instance storage back into memory and is resumed.
- **Provider callback** A call from the workflow instance to an application-defined workflow provider (for example, to resolve users for assignment or to calculate a due date). Workflow providers are integration points for developers to inject custom code for resolving users, due

dates, user hierarchies, or queues. A *provider callback* is a synchronous call from the workflow instance back into an X++ workflow provider.

- **Infrastructure callback** A call from the workflow instance back into X++ to perform infrastructure-related activities. One example is to create work items for each user returned from a call to a participant provider.

Figure 8-9 shows the logical workflow interactions for approvals.

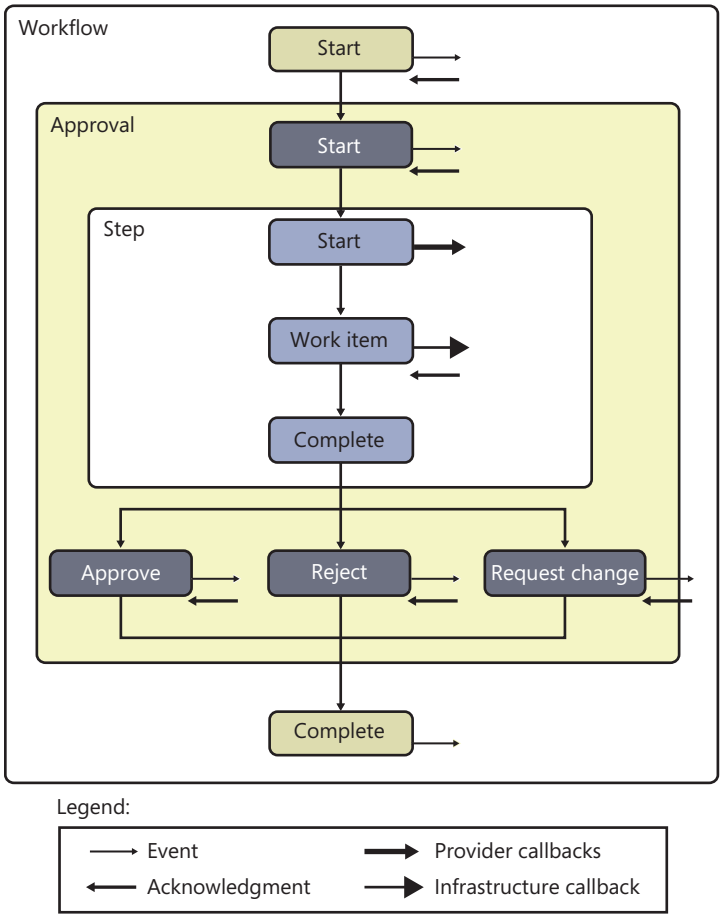


FIGURE 8-9 Logical approval workflow interactions.

In Figure 8-9, the outermost box represents the workflow itself. Nested inside are the approval (element) and within that, a single step. (An approval can contain multiple steps.) The smaller rectangular boxes represent events or outcomes. The symbols in the legend represent the four interaction types, which are positioned in Figure 8-9 where that type of interaction occurs. When the workflow starts, an event and an acknowledgment occur. Acknowledgments confirm that the workflow runtime received and processed a preceding event. A similar event and acknowledgment occur for the start of the approval element. When a step starts, callbacks invoke the workflow

providers to resolve the users for assignment and the due dates for the corresponding work items. The work items are then created through an infrastructure callback, and the workflow instance waits for the corresponding acknowledgments from the work items. Acknowledgments for work items are triggered when users take action on their assigned work items. After the step (or steps) complete, the outcome is determined based on the completion policies of the step, and the corresponding event is raised for that outcome. The workflow instance then waits for acknowledgment that the workflow runtime has processed the event that is associated with the outcome. Finally, the completion of the workflow itself raises an event.

Task interactions are similar to approvals, except that no steps and outcomes are unique for each task. Figure 8-10 shows the logical workflow interactions for tasks.

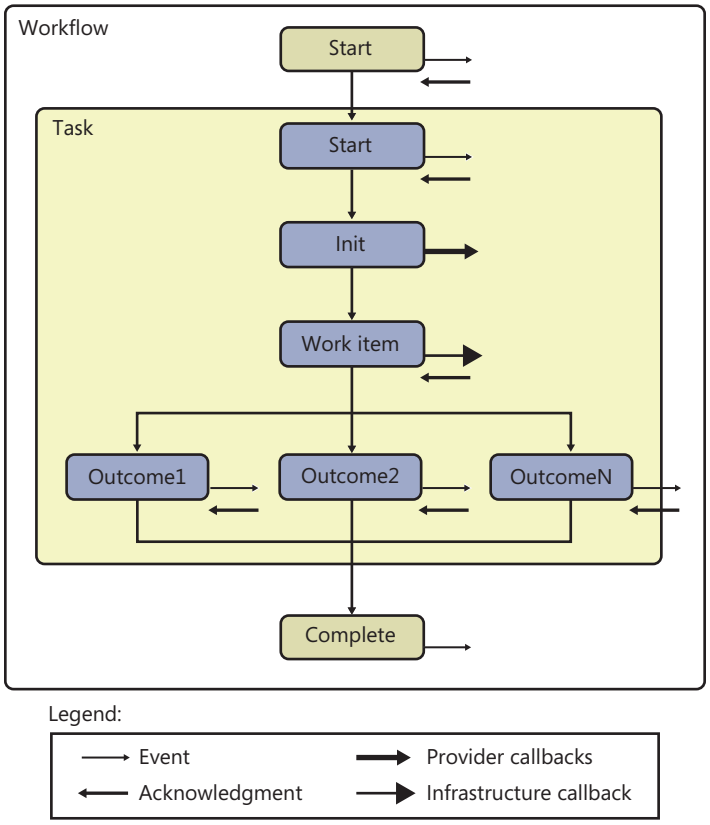


FIGURE 8-10 Logical task workflow interactions.

Workflow life cycle

This section describes the workflow process improvement life cycle, shown in Figure 8-11, and explains the implementation aspects of the life cycle in detail.

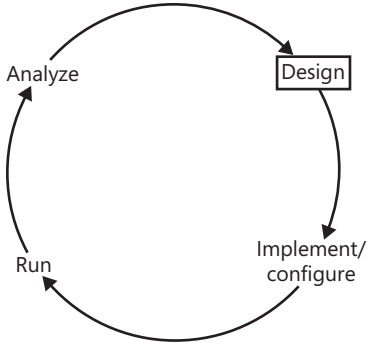


FIGURE 8-11 The workflow life cycle in Microsoft Dynamics AX 2012.

The workflow life cycle has four phases:

- **Design** Business process owners use their understanding of the organization to decide which parts of a business process that traverses Microsoft Dynamics AX 2012 need to be automated and then design a workflow to achieve this automation. They can collaborate with developers in this phase, or they might just communicate the workflow requirements to the developers.
- **Implement and configure** Developers implement workflow artifacts in Microsoft Dynamics AX 2012 based on the design of the business process. Business process owners then model the workflow using the graphical workflow editor. If this work is carried out on a test system, after successfully testing the workflow, the system administrator deploys the related artifacts and workflows to the live, or production, system.
- **Run** Users interact with Microsoft Dynamics AX 2012 as part of their day-to-day work, and in the course of doing so, might submit workflow documents to the workflow for processing or interact with workflows that are already activated.
- **Analyze** Business process owners evaluate the performance of the workflows that have been designed, implemented, and executed using the workflow analytical cube and performance reports introduced in Microsoft Dynamics AX 2012 to determine if any further changes are warranted.

This cycle is therefore repeated when a workflow that has been designed, implemented and configured, and deployed has to change in some way. Aside from performance, a change might result from a change in the business process or in the organization.

Implementing workflows

You can use the Microsoft Dynamics AX 2012 workflow infrastructure to automate aspects of a business process that are part of a larger business process automation effort. There is no single, correct approach to this undertaking. However, at a high level, you can follow the steps listed here to figure out and understand your existing business processes, determine how these business processes should function, and finally, to automate them by using workflow.

1. Map out existing business processes. This effort is often referred to as developing the *as-is* model and may involve the use of a business process modeling tool.
2. Analyze the as-is model to determine whether obvious improvements can be made to existing processes. These improvements are represented in another business process model, which is often referred to as the *to-be* model.
3. Design the way in which you're going to implement the to-be business process model—or the changes to the as-is model suggested by the to-be model. In this step, you might decide which parts of the to-be business process should be automated with workflow and which parts should remain manual.
4. For the parts of the business process model in which workflow is going to be used—and for the parts you want to automate—define the workflow document and then design one or more workflows. This step centers on the workflow document that the workflow will act on.
5. Implement the building blocks for the workflows, such as the business logic, and enabling workflow in the Microsoft Dynamics AX client, Enterprise Portal, or both.
6. Configure and enable the workflows, causing workflow instances to be created when a record for the workflow document is submitted.

The major advantage of the workflow infrastructure in Microsoft Dynamics AX 2012 is that it provides a significant amount of functionality out of the box, meaning that you don't have to write custom workflows. Businesses and organizations have more time to focus on improving their processes instead of writing and rewriting business logic.

Create workflow artifacts, dependent artifacts, and business logic

As a developer, once you understand the workflow requirements that the business process owner provides, you must create the corresponding workflow artifacts, dependent workflow artifacts, and business logic. You create these in the AOT by using the Microsoft Dynamics AX 2012 client. You write the business logic in X++.

Table 8-1 lists each workflow artifact and the steps you need to perform when creating it. The artifacts are listed in order of dependency.

TABLE 8-1 Workflow artifacts.

Artifact	Steps
Workflow category	1. Define the module in which the workflow type is enabled. For more information, see the section "Key workflow concepts," earlier in this chapter, and "How to: Create a Workflow Category" at *http://msdn.microsoft.com/en-us/library/cc589698.aspx*.
Approval	1. Define the approval workflow document. 2. Define approval event handlers for *Started* and *Canceled*. 3. Define approval menu items for *Document*, *DocumentWeb*, *Resubmit*, *ResubmitWeb*, *Delegate*, and *DelegateWeb*.

Artifact	Steps
	4. Enable or disable approval outcomes. 5. Define approval outcome menu items for *Action* and *ActionWeb*. 6. Define an approval outcome event handler. 7. Define the *DocumentPreviewFieldGroup*. For more information, see "How to: Create a Workflow Approval" at *http://msdn.microsoft.com/en-us/library/cc596847.aspx*.
Task	1. Define the task workflow document. 2. Define task event handlers for *Started*, *Canceled* and *WorkItemCreated*. 3. Define task menu items for *Document*, *DocumentWeb*, *Resubmit*, *ResubmitWeb*, *Delegate*, and *DelegateWeb*. 4. Add or remove task outcomes. 5. Define task outcome menu items for *Action* and *ActionWeb*. 6. Define task outcome event handler. 7. Define the *DocumentPreviewFieldGroup*. For more information, see "How to: Create a Workflow Task" at *http://msdn.microsoft.com/en-us/library/cc601939.aspx*.
Automated Task	1. Define the automated task workflow document. 2. Define automated task event handlers for *Execution* and *Canceled*. For more information, see "Walkthrough: Adding an Automated Task to a Workflow" at *http://msdn.microsoft.com/en-us/library/gg862506.aspx*.
Workflow type	1. Define the workflow document. 2. Define event handlers for workflow *Started*, *Completed*, *ConfigDataChanged*, and *Canceled*. 3. Define menu items for *SubmitToWorkflow*, *SubmitToWorkflowWeb*, *Cancel*, and *CancelWeb*. 4. Define the workflow category. (Select a category from the existing categories.) 5. Define supported approvals, tasks, and automated tasks (these will then be displayed in the graphical workflow editor.) 6. Enable or disable activation conditions for workflows based on the type. For information about creating workflow types, see "Walkthrough: Creating a Workflow Type" at *http://msdn.microsoft.com/en-us/library/cc641259.aspx*.

Table 8-2 identifies the *dependent* workflow artifacts that are referenced in Table 8-1.

TABLE 8-2 Dependent workflow artifacts.

Dependent workflow artifact	Description
Workflow document query	This query defines the data in Microsoft Dynamics AX 2012 that a workflow acts on and exposes certain fields that the business process owner uses for constructing conditions in the graphical workflow editor. The query is defined under the *Queries* node in the AOT and it is required for all workflows.
Workflow document class	This X++ class references the workflow document query and any calculated fields to be made available when constructing conditions. This class is created under the *AOT\Classes* node and extends the *WorkflowDocument* base class. This class is required because workflow types and elements must bind to a workflow document class. For information about derived data, see the section "Key workflow concepts," earlier in this chapter.
SubmitToWorkflow class	This X++ class is the menu item class for the *SubmitToWorkflow* menu item that displays the Submit To Workflow dialog box in the Microsoft Dynamics AX 2012 user interface. The Submit To Workflow dialog box allows the user to enter comments associated with the submission. A *SubmitToWorkflow* class then activates the workflow. If state is being managed in the record that has been submitted to workflow, this class can be used to update the state of the record. This class is created under the Classes node of the AOT.

Dependent workflow artifact	Description
State model	A defined set of states and state transitions (supported changes from one state to another) used to track the status of workflow document records during their life cycle. For example, a document can have the following states: *Not Submitted*, *Submitted*, *ChangeRequested*, or *Approved*. There is currently no state model infrastructure in Microsoft Dynamics AX, so you must implement any state model that is required. For more information, see the section "State management," later in this chapter.
Event handlers	Event handler code consists of business logic that is written in X++ and then referenced in the workflow type, the approval element, approval outcomes, the task element, task outcomes, and the automated task element. If a workflow document has an associated state model, you must write event handler code to transact workflow document records through the state model when being processed by using workflow. Event handler X++ code is created under the AOT\Classes node.
Action and display menu items	For information about menu items, see the section "Key workflow concepts," earlier in this chapter. Both types of menu item are created under the *AOT\Menu* Items or *AOT\Web\Web Menu Items* node. For more information about the menu items used in the workflow infrastructure, see "How to: Associate an Action Menu Item with a Workflow Task or Approval Outcome" (*http://msdn.microsoft.com/en-us/library/cc602158.aspx*) and "How to: Associate a Display Menu item with a Workflow Task or Approval" (*http://msdn.microsoft.com/en-us/library/cc604521.aspx*).
Custom workflow providers	If the functionality of the workflow providers included with Microsoft Dynamics AX 2012 isn't adequate for a given set of requirements, you can develop your own workflow provider. Custom workflow provider X++ classes are created under the *AOT\Classes* node and then referenced in one or more providers (under the *AOT\Workflow\Providers* node). For more information about workflow providers, including where they are used, see "Workflow Providers Overview" at *http://msdn.microsoft.com/en-us/library/cc519521.aspx*.
canSubmitToWorkflow method	This method is required to inform the workflow common UI controls that the record in the form is ready to be submitted to the workflow. While in Microsoft Dynamics AX 2009, the form *canSubmitToWorkflow* method was overridden, in Microsoft Dynamics AX 2012, this logic can be implemented on the table's *canSubmitToWorkflow* method instead.

State management

A *state model* defines a set of states and the transitions that are permitted between the states for a given record type, along with an initial state and a final state. State models exist to provide a prescriptive life cycle for the data they are associated with. The current state value is often stored in a field on a record. For example, the PurchReqTable table (the header for a purchase requisition) has a *status* field that is used to track the approval state of a purchase requisition. The business logic for purchase requisitions is coded to respect the meaning of each state and the supported state transitions so that a purchase requisition record can't be converted into a purchase order before the state is approved.

The simplest way to add and manage the state on a record is to use a single field to store the current state, but you have to determine the approach that makes the most sense. You would then create a static X++ class that implements the business logic that governs the state transition. Conceptually, you can think of this class as a *StateManager* class. All existing business logic that

performs the state transitions should be refactored to use this single, central class to perform the state transitions, in effect isolating the state transition logic into a single class. From a workflow perspective, state transitions always occur at either the beginning or the conclusion of a workflow element. This is why all workflow tasks and workflow approvals have *EventHandler*s that can be used to invoke a *StateManager* class. Figure 8-12 shows the dependency chain between an event handler and the workflow document state.

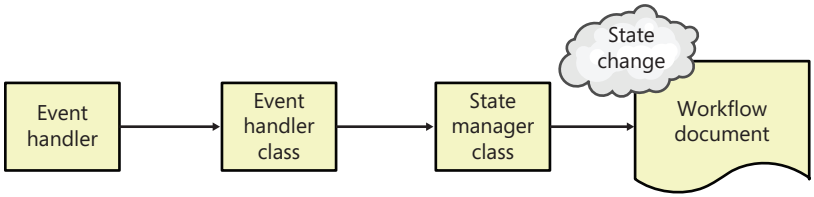


FIGURE 8-12 State management dependency chain.

When you decide to enable a workflow for a table in Microsoft Dynamics AX 2012 and determine that the table has a state that must be managed, you *must* refactor all business logic to respect the state model that you define to avoid unpredictable results. Create operations should always create a record with the *initial* state (for the state model). Update operations must respect the current state and fail if the state isn't as expected. For example, it shouldn't be possible to change the business justification of a purchase requisition after it has been submitted for approval. Managing the state of the record during each update so that the current state is verified and the next logical state is updated is typically implemented in the update method on the table by calling the *StateManager* class. If it returns a value of *true*, perform the update. If not, throw an exception and cancel the operation. Figure 8-13 shows a simple state model for a record.

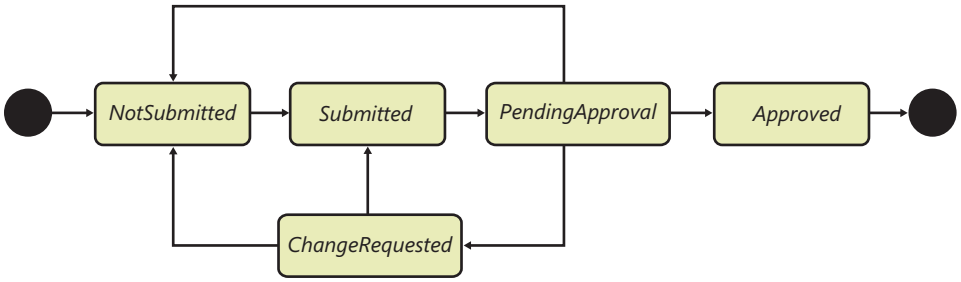


FIGURE 8-13 A simple state model for approvals.

In Figure 8-13, the initial state is *NotSubmitted*. When a record is submitted to workflow, the state changes to *Submitted*. After the workflow is activated, the state becomes *PendingApproval*. If a workflow participant selects the Request Change action, the state changes to *ChangeRequested*. After all approvals are submitted, the final state is *Approved*.

Create a workflow category

You use workflow categories to associate a workflow type with a module. This association restricts the list of types that are shown when the business process owner edits a workflow for a particular module, preventing a list of all workflow types from being displayed. For example, if a user is in the Accounts Payable module, the user sees only the workflow types that are bound to Accounts Payable. The mechanism behind this grouping is a simple metadata property on the workflow type called *Workflow category*. This property allows you to select an element from the module *enum* (AOT\Data Dictionary\Base enums\ModuleAxapta).

With this mechanism, it is easy for ISVs and partners who create their own modules to extend the module *enum* and thus have workflow types that can be associated with that module. Note that a workflow category can be associated with only one module.

Create the workflow document class

The purpose of a workflow is to automate all or part of a business process. To do this, it must be possible to define various rules for the document that is being processed by workflow. In Microsoft Dynamics AX 2012, these rules are called *conditions*. A business process owner creates conditions when modeling the workflow. For example, conditions can be used to determine whether a purchase requisition is approved automatically (without any human intervention). Figure 8-14 shows a simple condition defined in the graphical workflow editor.

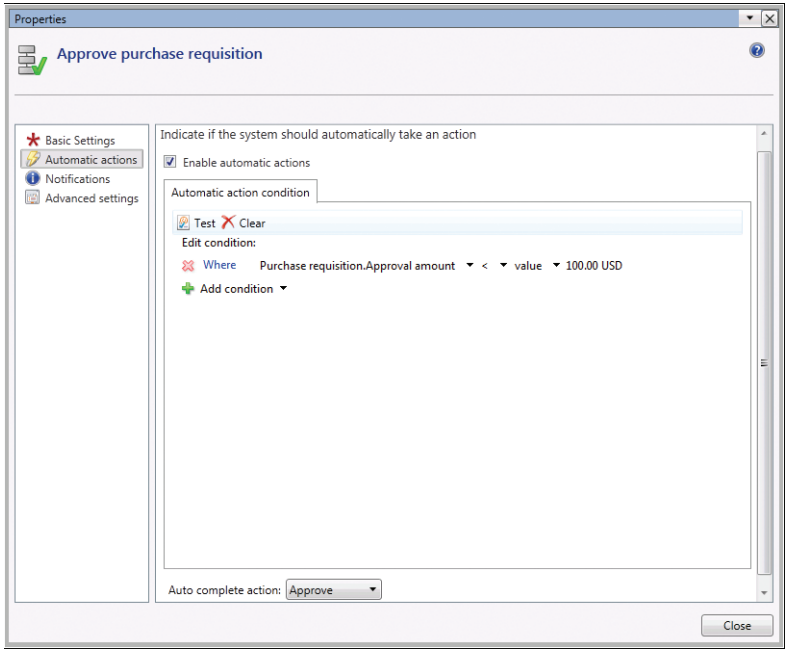


FIGURE 8-14 A simple condition defined in the graphical workflow editor.

When a business process owner defines a condition by using the graphical workflow editor, he or she needs to make sure that users have a way to select the fields from the workflow documents they want to use. On the surface, this seems simple, but two requirements complicate the task. First, not all the fields in a table might make sense to the business process owner, and therefore only a subset of the fields should be exposed. Second, it must be possible to use calculated fields (also called *derived data*). The workflow document class meets these two requirements by functioning as a thin wrapper around an AOT query that defines the available fields and by providing a mechanism for defining calculated fields.

The AOT query enables developers to define a subset of fields from one or more related tables. By adding nested data sources in a query, you can model complex data structures. However, the most common usage is to model a header-line pattern. At design time, when the business process owner is editing a workflow, the AOT query is used by the condition editor to determine which fields to display to the business process owner.

The workflow infrastructure uses a prescriptive pattern to support calculated fields by using *parm* methods that are defined within the workflow document class. These methods must be prefixed with *parm* and must implement a signature of *(CompanyId, TableId, RecId)*. The workflow infrastructure then, at run time, calls the *parm* method and uses the return value in the condition evaluation. This design enables developers to implement calculated fields in *parm* methods on the workflow document class.

Note When the expression builder constructs the list of fields, it uses the labels for the table fields as the display names for the fields. The display name for a calculated field is defined by the extended data type label of the return types. For *enums*, this is defined by the *enum* element label.

Creating a workflow document class involves creating an X++ class that extends *WorkflowDocument*. You must override the *getQueryName* method to return the name of the workflow document query. Figure 8-15 shows a sample X++ class that extends *WorkflowDocument*.

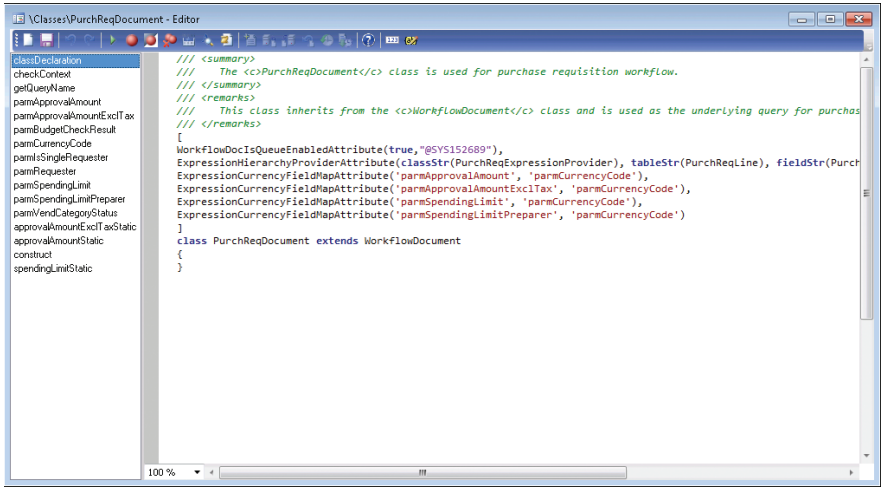


FIGURE 8-15 A sample X++ class that extends from the workflow document.

Creating a *parm* method involves adding a method to the workflow document class and then adding X++ code to calculate or otherwise determine the value to be returned, as shown in Figure 8-16.

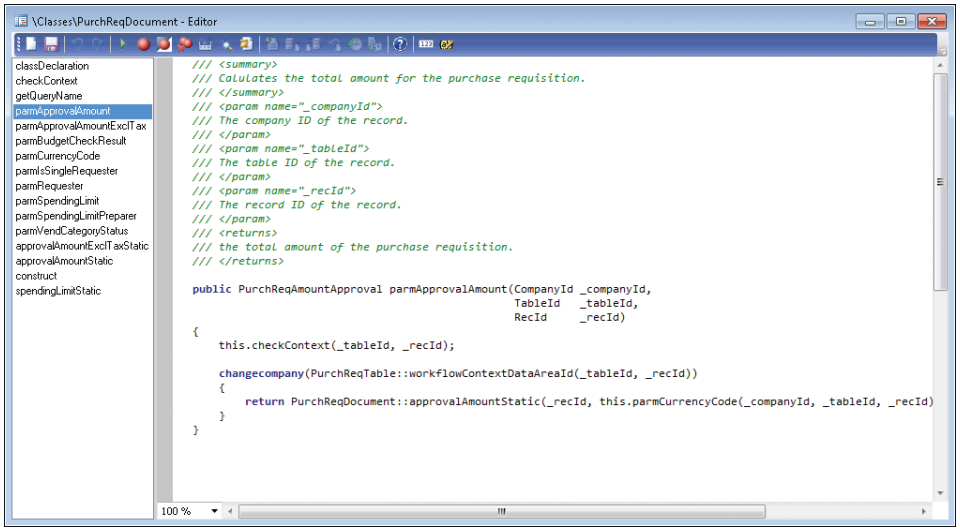


FIGURE 8-16 A *parm* method within a workflow document class that returns the approval amount (which is calculated).

Add a workflow display menu item

Workflow display menu items enable users to navigate directly to the Microsoft Dynamics AX 2012 client form (or Enterprise Portal webpage) from which they can select one of the available workflow actions. A user is prompted to participate in a workflow when he or she receives a work item from the workflow at run time. When viewing the work item, the user can click Go To <*Label*>. This button is automatically mapped to the workflow display menu item, and the button text (<*Label*>) is the label of the root table of the workflow document query.

This design enables developers to create task-based forms that are focused on the particular task at hand, rather than having to create monolithic forms that assume the user knows where in the process he or she is acting and which fields and buttons to use.

Activate the workflow

Workflows in Microsoft Dynamics AX 2012 are always explicitly activated; either a user does something in the Microsoft Dynamics AX 2012 client or in Enterprise Portal that causes workflow processing to start, or the execution of business logic starts a workflow. (Once you understand how users activate a workflow, you can use this knowledge to activate workflows through business logic.)

For the first activation approach to work, the workflow infrastructure must have a way to communicate information to the user about what to do. For example, it might be relevant to instruct the user to submit the purchase requisition for review and approval at the appropriate time. The requirements to communicate with users throughout the workflow life cycle gave Microsoft an opportunity to standardize the way users interact with workflow in both the Microsoft Dynamics AX 2012 client and Enterprise Portal, including activating a workflow, and this resulted in the development of *workflow common UI controls*. The workflow common UI controls include the yellow workflow message bar (highlighted in Figure 8-17) and the workflow action button, labeled Submit.

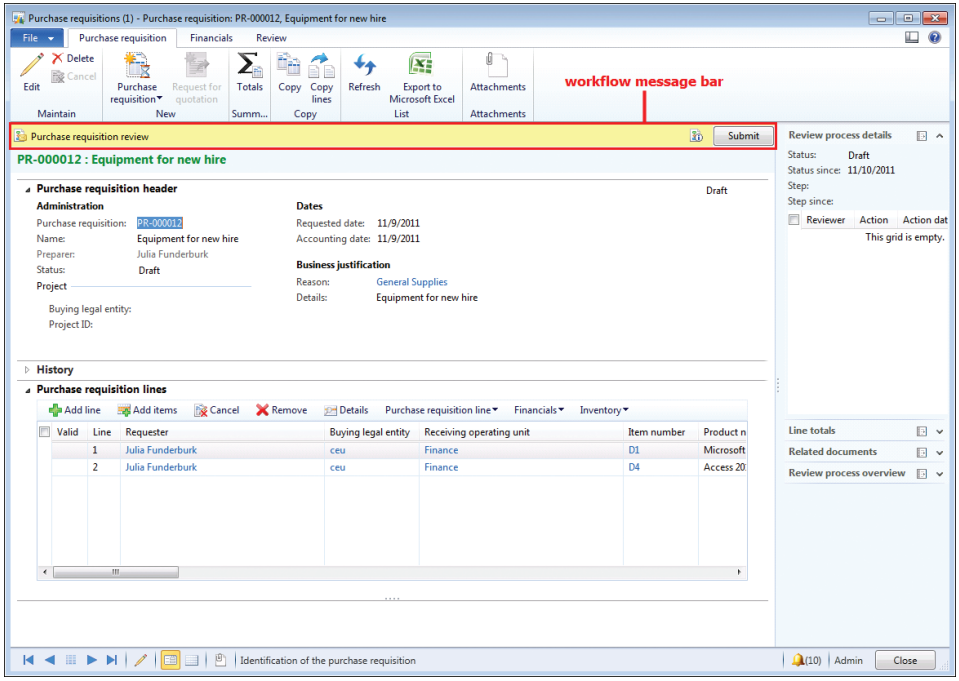


FIGURE 8-17 A purchase requisition ready to be submitted to workflow for processing.

The workflow common UI controls appear on the Purchase requisition form because that form has been enabled for workflow. To enable workflow in a form, you set the *WorkflowEnabled* property on the form to Yes in the Properties window, which is shown in Figure 8-18. You must also set the *WorkflowDataSource* property to one of the data sources on the form. The selected data source must be the same as the root data source that is used in the query referenced by the workflow document. Finally, you can set the *WorkflowType* property to constrain the form to use a specific workflow type.

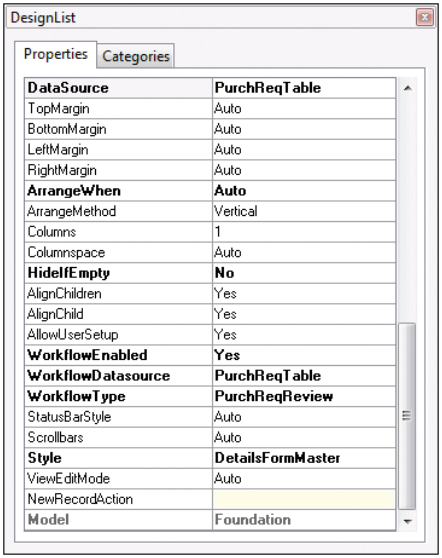


FIGURE 8-18 Design properties for a Microsoft Dynamics AX 2012 form, including those for workflow.

If workflow is enabled for a form, the workflow common controls automatically appear in three cases:

- When the currently selected document can be submitted to workflow (the canSubmitToWorkflow table or form method returns *true*)
- When the current user is the originator of a workflow that has acted on the currently selected document
- When the current user has been assigned to a work item for which he or she must take an action

The workflow common control uses the algorithm shown in Figure 8-19 to determine which workflow to use.

After a workflow has been identified, it's easy for the workflow common UI controls to obtain the *SubmitToWorkflow* action menu item. This action menu item is then dynamically added to the form, along with the yellow workflow message bar.

If you look at the *SubmitToWorkflow* action menu item for the *PurchReqApproval* workflow type, you'll notice that it is bound to the *PurchReqWorkflow* class. When you click the Submit button, the action menu items call the *main* method on the class it is bound to; thus, the code that activates the workflow is called from the *main* method. In this case, the call to the workflow activation API has been isolated within the *submit* method.

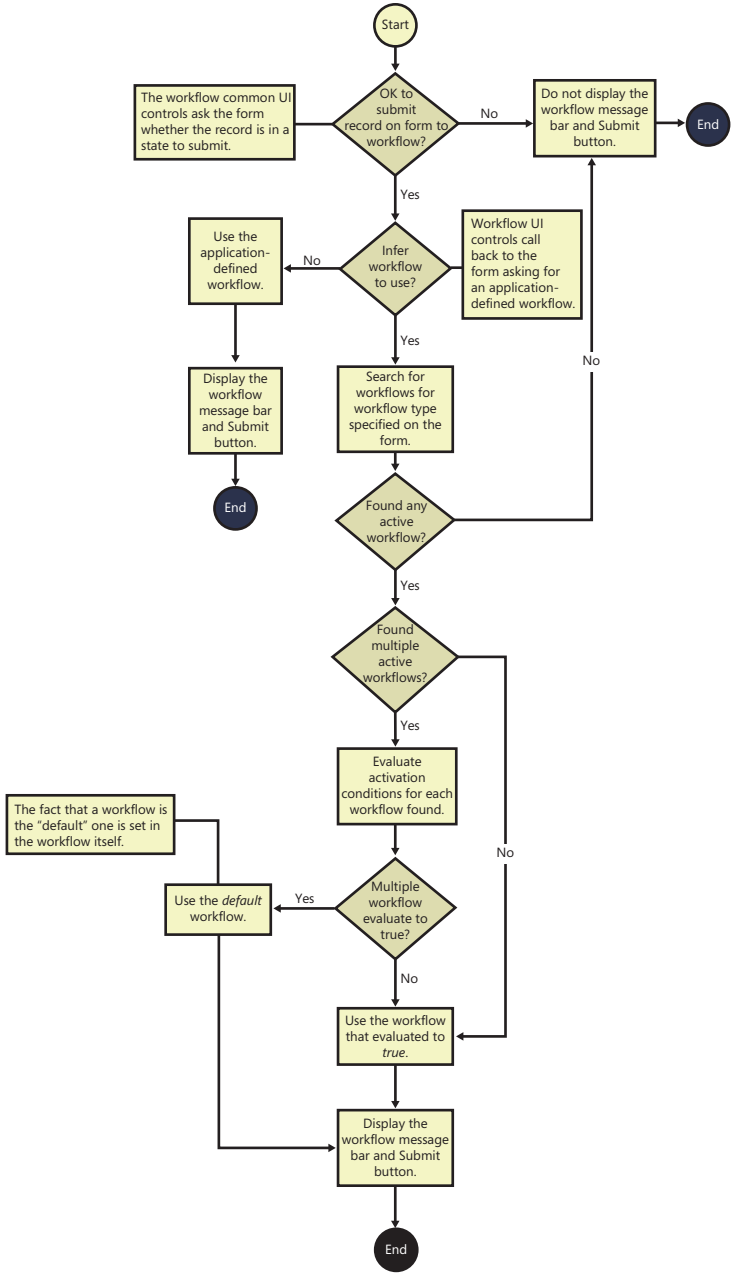


FIGURE 8-19 Workflow activation logic flowchart.

In Figure 8-20, notice how the *Workflow::activatefromWorkflowType* method is used. You can use two additional APIs to activate workflows: *Workflow::activatefromWorkflowConfiguration* and *Workflow::activateFromWorkflowSequenceNumber*.

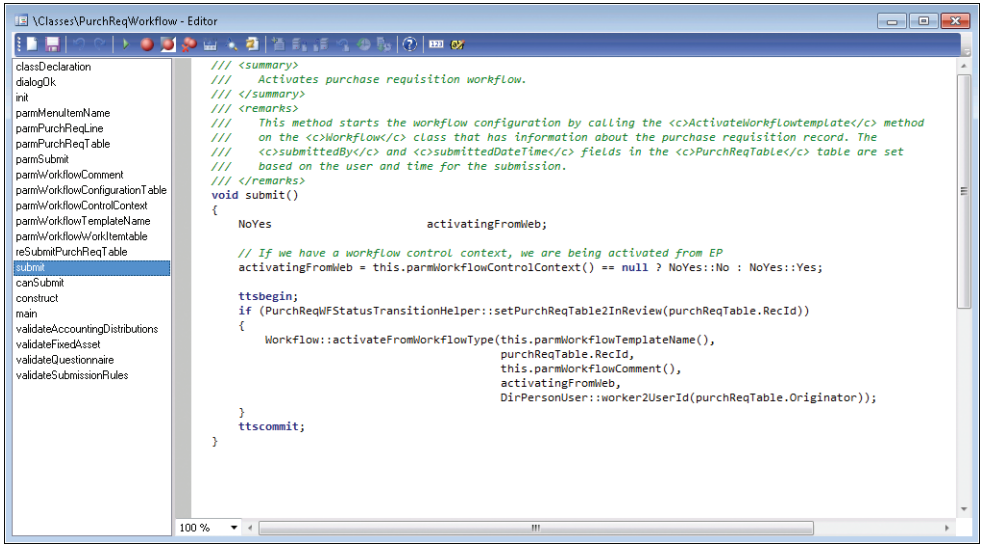


FIGURE 8-20 The *Submit* method for the purchase requisition workflow.

For information about how to use these APIs, see the Microsoft Dynamics AX 2012 developer documentation on MSDN: *http://msdn.microsoft.com/en-us/library/cc586793.aspx*.

Understanding how to activate a workflow is important, but it is equally important to understand how to prevent a workflow from being activated. For example, you don't want a user to submit a record to workflow before it is in a state to be submitted. An override method on the table or form, *canSubmitToWorkflow*, addresses this requirement. The *canSubmitToWorkflow* method returns a Boolean value. A value of *true* indicates that the record can be submitted to workflow. When the workflow data source on the form is initialized or when the record changes, this method is called; if it returns *true*, the Submit button is enabled. Typically, you should update the state of the document after invoking the workflow activation API so that you can correctly denote whether a document has been submitted to workflow. (In Figure 8-20, the purchase requisition is transitioned to the *In Review* state.)

Note If the *canSubmitToWorkflow* method hasn't been overridden either at the table or form level, the workflow common UI controls won't appear, leaving a reserved space at the top of the form usually occupied by the controls.

CHAPTER 9

Reporting in Microsoft Dynamics AX

In this chapter

Introduction 275
Inside the Microsoft Dynamics AX 2012 reporting framework 276
Plan your reporting solution 279
Create production reports 281
Create charts for Enterprise Portal 289
Troubleshoot the reporting framework 296

Introduction

Reporting is critical for any organization because it is a primary way that users gain visibility into the business. Reports help users understand how to proceed in their day-to-day work, make more-informed decisions, analyze results, and finally take action. Microsoft Dynamics AX 2012 provides a variety of reporting tools that developers can use to create appealing and useful reports for both the Microsoft Dynamics AX Windows client and the Microsoft Dynamics AX Enterprise Portal web client.

Microsoft Dynamics AX 2012 and Microsoft Dynamics AX 2012 R2 have introduced some important enhancements to the Microsoft Dynamics AX reporting framework.

Microsoft SQL Server Reporting Services (SSRS) was introduced in Microsoft Dynamics AX 2009. In Microsoft Dynamics AX 2012, the SSRS reporting framework has become the primary reporting engine for Microsoft Dynamics AX. The SSRS platform provides customers with access to an expanded pool of resources, including developers, partners, and documentation to support this standard industry solution.

Note Microsoft Dynamics AX 2012 continues to offer the MorphX platform as a fully integrated solution and to allow customers enough time to transition their existing reporting solutions to the SSRS framework.

Microsoft Dynamics AX 2012 also offers enhanced integration with Microsoft Visual Studio 2010. New Visual Studio report templates are available for Microsoft Dynamics AX, and you can use Visual

Studio to create both auto-design and precision design reports more easily. The Enterprise Portal (EP) Chart Control is a new chart data visualization tool introduced in the Microsoft Dynamics AX 2012 R2 release. This tool provides a high-performance alternative to SSRS reports in Role Centers and other pages in Enterprise Portal. The EP Chart Control is the recommended solution for interactive presentations for large volumes of data in Enterprise Portal. This new utility is an extension of the ASP.Net Chart Control and provides access to all of its underlying functions, including 35 distinct chart types. The EP Chart Control provides automatic element formatting to make charts look appealing and offers declarative solutions for accessing Microsoft Dynamics AX data that is captured in both online analytical processing (OLAP) and online transaction processing (OLTP) databases, simplifying the developer experience.

This chapter focuses on using Visual Studio to create SSRS reports for the Microsoft Dynamics AX client and charts for EP. For a complete list of new developer features for this release, see "What's New: Reporting for Developers in Microsoft Dynamics AX 2012," at *http://msdn.microsoft.com/en-us/library/gg724100.aspx*.

Inside the Microsoft Dynamics AX 2012 reporting framework

This section compares client-side and server-side reporting solutions, and provides insights into how the Microsoft Dynamics AX reporting framework offers seamless integration that enables easy access to OLTP data and aggregated data that is managed in SQL Server Analysis Services (SSAS). This section also identifies the key components of the Microsoft Dynamics AX 2012 reporting framework and describes their functions.

In the realm of reporting, there are two primary architectures to compare when considering a solution: client-side and server-side. Briefly stated, client-side reporting uses the power of the client to carry the bulk of the load when reports are constructed. The MorphX reporting framework is an example of a client-side reporting solution. For the most part, server requests are made simply to access the data. Server-side reporting, on the other hand, uses various server resources to aid in the processing and construction of a report. The Microsoft Dynamics AX 2012 reporting framework is a server-side reporting solution. As you might expect, there are many trade-offs between the two models. The next sections discuss some of the benefits and limitations that are associated with each design.

Client-side reporting solutions

As mentioned earlier, the MorphX framework is a proprietary client-side solution that is fully integrated into the Microsoft Dynamics AX integrated development environment (IDE). In this model, reports contain references to data sources that are bound to local Microsoft Dynamics AX tables and views. They also define the business logic.

Figure 9-1 illustrates the architecture of a client-side reporting solution.

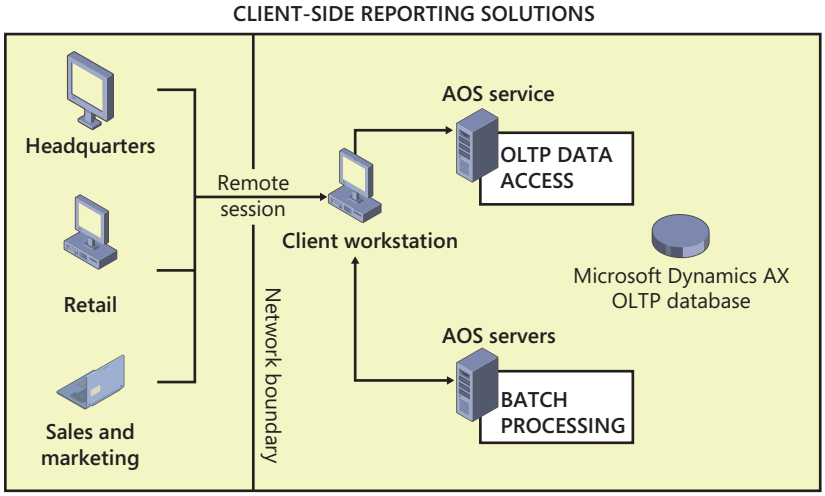


FIGURE 9-1 A client-side reporting solution.

The key benefits of a client-side reporting solution include the following:

- Business logic is executed along with the design definition, allowing for programmable sections in reports.
- No deployment is needed: you import the report, and it's immediately available to the client.
- X++ developers can use familiar tools to construct report designs.

Notable disadvantages of a client-side reporting solution include the following:

- Client components must be installed for a user to be able to view a report.
- Users outside the domain, (outside the network boundary shown in Figure 9-1), must connect to a Microsoft Dynamics AX client through Remote Desktop Connection to access reports.
- Access is limited to the data that is accessible from the client.
- Components such as business logic, parameter management, and designs cannot be shared across reporting solutions.

Server-side reporting solutions

SSRS, the primary reporting platform for Microsoft Dynamics AX, is a server-side reporting solution. This framework takes advantage of an industry solution that offers comprehensive reporting functionality for a variety of data sources. This platform includes a complete set of tools that you can use to create, manage, and deliver reports. With SSRS, you can create interactive, tabular, graphical, or free-form reports from relational, multidimensional, or XML-based data sources.

Figure 9-2 illustrates the architecture of a generic server-side reporting solution.

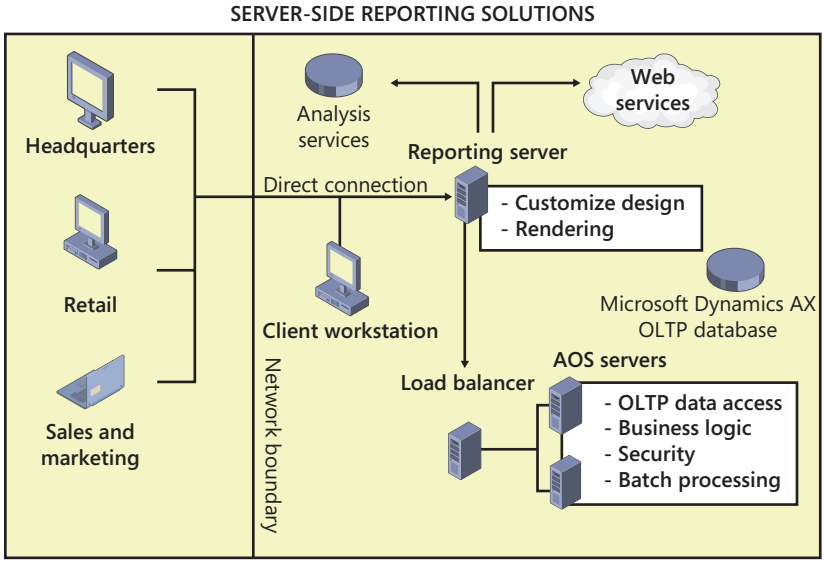


FIGURE 9-2 A server-side reporting solution.

The key benefits of a server-side reporting solution are as follows:

- It provides access to external data sources, including SSAS and web services.
- It supports reporting in thin clients, with no additional client components required. Users outside the domain (shown as the network boundary in Figure 9-2) can connect to Enterprise Portal to access reports, instead of having to connect remotely to the Microsoft Dynamics AX client, as in a client-side reporting solution.
- The workload for report rendering is performed on the server.
- Design caching improves the overall performance of report generation.

The key limitations of a server-side reporting solution are as follows:

- The lack of a direct connection to local printers affects some scaling scenarios.
- Report modifications must be deployed before they can be accessed by the client.
- It requires additional server management for system administrators.

Report execution sequence

Figure 9-3 illustrates the architecture of the Microsoft Dynamics AX 2012 reporting framework.

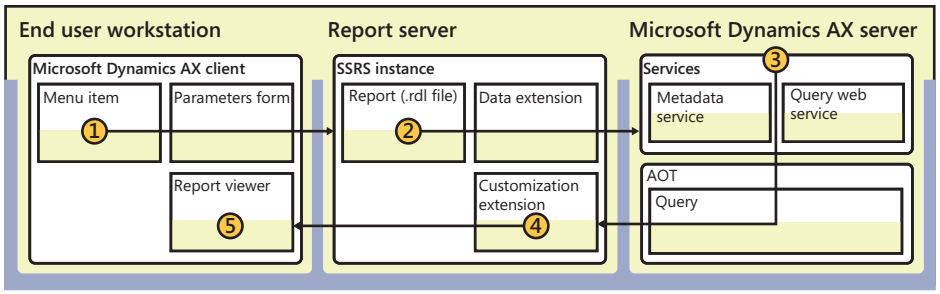


FIGURE 9-3 The Microsoft Dynamics AX 2012 reporting framework.

The following list corresponds to the numbered items in Figure 9-3:

1. **Menu item** An entry point into the report execution sequence. Menu items contain predefined hyperlinks that are used to instantiate and execute reports. Configuration keys can be linked to menu items to manage user access.
2. **Report definition (.rdl file)** An XML representation of an SSRS report definition, containing both the data retrieval and design layout information for a given report.
3. **Application Object Server (AOS)** The core of the Microsoft Dynamics AX server platform. The query web service is used to access OLTP data.
4. **Customization extension** Design customizations are applied to produce a personalized view of the report.
5. **Report viewer** The report is rendered for the user in the client.

Plan your reporting solution

Applying a well-thought-out design will greatly simplify the development process and ongoing task of maintaining your report. This requires planning based on your unique set of report requirements.

Reporting and users

You can create two types of reports in Microsoft Dynamics AX: production reports, which present data that is predefined, and ad hoc reports, which present data that is selected by users. When planning out your reporting solution, ask yourself the following questions:

- Who are the users of the report, and what are their roles within the business?
- What information do the users need to complete their tasks?
- How do the users want to respond to the information that is presented?

You can categorize reporting functions on two axes: data depth and business activity. As shown in Figure 9-4, the roles that users play in an organization and their unique reporting requirements fall at one point (or perhaps several points) on these axes.

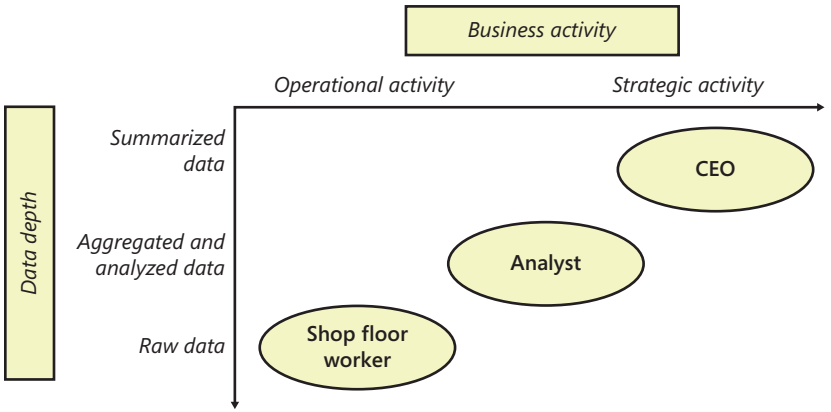


FIGURE 9-4 An illustration of how users in various business roles work with different views of business data that require different kinds of reports.

Here are some details about the reporting needs of the roles shown in Figure 9-4:

- The CEO, who is interested in monitoring the health of the business, periodically uses strategic reports that provide summarized views of data across time periods.
- The analyst examines the business, looking for patterns that might lead to a change in business plans and priorities. Analysts rely on reports that allow the data to be interactive, so that data can be sliced by department or region. They also value visuals that simplify the process of detecting patterns and trends that may feed into the CEO's decisions.
- The shop floor worker is primarily concerned with the day-to-day activities of the business and uses reports that reflect the immediate needs of his or her area. An inventory list is a simple type of report that the shop floor worker finds great value in.

Roles in report development

The role of report developer can literally be split into two distinct functions:

- **Constructing the report dataset** This task consists of identifying all data elements that are either visualized in the report dataset or used to support user interactions. This task is well suited to developers who are familiar with the MorphX development environment and the structure of the customer's business data.
- **Defining the report design** Authoring report designs requires familiarity with the report design experience provided by Visual Studio 2010.

Dividing these tasks among more than one individual is ideal because it encourages a clear separation between the business logic and the presentation layer.

Figure 9-5 provides a high-level view of the report development process.

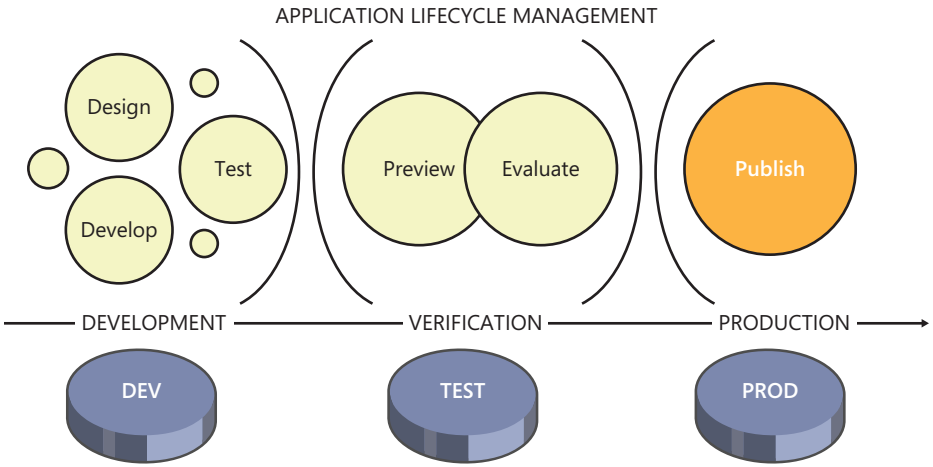


FIGURE 9-5 The report development process.

Traditionally, reports are developed in a contained environment that is shared by a team of developers. When the developer feels that the solution satisfies the reporting requirements, he or she uses the Visual Studio tools to publish the report in the DEV, or development environment, for verification from within the client. When the developer is satisfied, the reporting project is packaged as a model or project and moved into the TEST environment. This is where the new report is put to the test in a simulated production environment, to ensure that both the functionality and performance are sound. Finally, the report is published to the PROD, or production environment, so that it is accessible to the designated set of users.

To learn more about creating a report for Microsoft Dynamics AX by using Visual Studio 2010, see the detailed step-by-step instructions in the reports section of the Microsoft Dynamics AX 2012 SDK at *http://msdn.microsoft.com/en-us/library/cc557922.aspx*. These topics have comprehensive descriptions for all the core scenarios that report developers are likely to encounter.

Create production reports

You use Visual Studio 2010 to create and modify Microsoft Dynamics AX SSRS reports. In Microsoft Dynamics AX 2012, the report development tools have been augmented to offer a fully integrated experience. These tools provide report designers the benefit of working with the familiar Visual Studio 2010 IDE and the ability to use the rich reporting features in SSRS.

The Microsoft Dynamics AX 2012 report development tools offer a model-based approach for creating reports that is based on fully customizable templates that define the layout and format of the reports.

The Microsoft Dynamics AX reporting development tools consist of a modeling tool, Model Editor, that you can use to visualize the report elements as you develop a report. The reports that you create are stored in the Report Definition Language (RDL) format specified by SSRS. By using this widely adopted format, you can take advantage of the many features (for example, charting, interactivity, and access to multiple data sources) that make SSRS a popular choice for production reports. You can store, deploy, manage, and process reports on the report server by using the integrated Visual Studio report development tools.

The Microsoft Dynamics AX reporting tools also include a new Visual Studio project template called Microsoft Dynamics AX Reporting Project. This new project type simplifies the process of creating SSRS reports that bind to data in Microsoft Dynamics AX.

The Dynamics AX Reporting Project template has the following features:

- It allows a report to retrieve Microsoft Dynamics AX data from the AOS by using either a Microsoft Dynamics AX query or a Report Data Provider object.
- It defines the report parameters and layout of the controls.
- It uses references to Microsoft Dynamics AX labels to produce localized strings based on the user's current Microsoft Dynamics AX language.
- It allows SSRS reports to be created and modified in the Application Object Tree (AOT).
- It can be used to deploy report customizations to the report server.

Model elements for reports

Three basic components make up any SSRS report: the controls, the design definition, and, of course, the data:

- The controls, often referred to as the *parameters* or *inputs*, can be either provided by the user or derived from the context of the session. For example, the reporting framework automatically selects the language for a report based on the user's settings in Microsoft Dynamics AX. Controls are used to select the design, alter the format and layout of the report, and influence the dataset that is ultimately rendered in the report.
- The design of the report contains a collection of elements, such as text boxes, tables, matrices, and charts that define the look and feel of the report. You construct the report design by using an augmented Visual Studio 2010 Report Designer experience.
- The data to be displayed in a report can be derived from a number of sources, including the Microsoft Dynamics AX OLTP database, SSAS, external databases, .NET service providers, and XML data files. Datasets are used to establish data connections to various sources, and they can be used interchangeably by one or more report designs.

Figure 9-6 illustrates an example of a Report model in Visual Studio 2010, showing the three components of an SSRS report. The following sections describe each component in more detail.

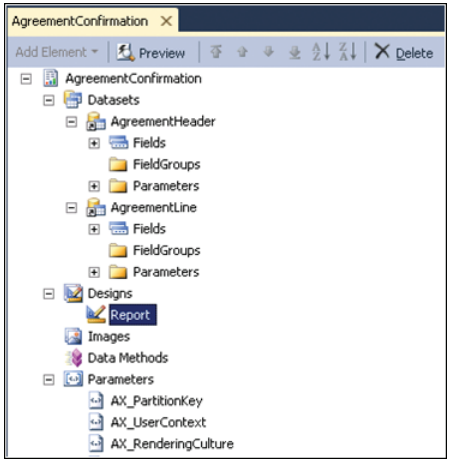


FIGURE 9-6 A Report model in Visual Studio.

Controls

Controls are used to filter the data that is displayed in a report, connect related reports, and control report presentation. For example, you can write an expression to change the font based on a parameter that is passed to the report. Design parameters can be directly bound to dataset controls or used in run-time evaluations that affect the report design. You use Model Editor to define the grouping and order of report parameters when a scenario is complex; for example, if you want to use multiple nested groups. The order in which the report parameters are listed in a group is the order that the user sees them on the report. This makes the grouping and order in Model Editor easy to see as you define the report. For more information, see "How to: Group and Order Report Parameters by Using Visual Studio" (*http://msdn.microsoft.com/EN-US/library/gg731925*).

Designs

A report design represents the layout of a report. A report can have multiple designs that share datasets and parameters. This is appropriate in scenarios where you have similar reports based on the same dataset. You can create the following types of report designs:

- **Auto design** A report design that is generated automatically based on the report data. You create an auto design report using Model Editor. The auto design functionality provides an efficient way to create the most common types of reports, such as a customer list or a list of inventory items. An auto design layout consists of a header, a body that contains one or more data regions, and a footer, as shown in Figure 9-7.

  You control the content that is displayed in each area in an auto design. For example, you can include a report title and the date in the header and display the page number in the footer, or you may not want to display anything in the header and footer.

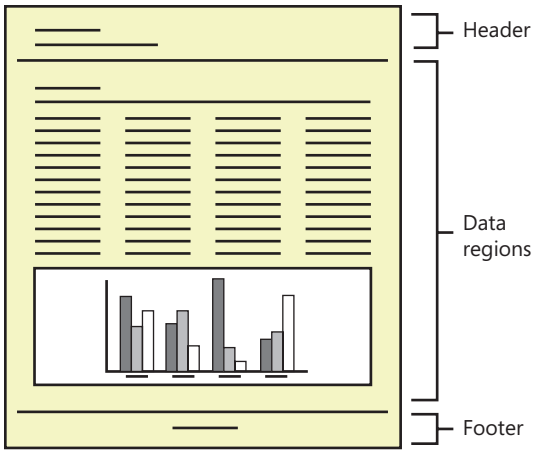


FIGURE 9-7 Auto design report layout.

The data regions that are displayed in an auto design depend on the datasets that you created when you defined the data for the report. When you define a dataset, you can specify the type of data region that will be used to render the data whenever the dataset is used in an auto design. Data can be displayed in table, list, matrix, or chart format. One way to create an auto design is to drag a dataset onto the node for the auto design in the model.

- **Precision design** A report design that you create by using SQL Server Report Designer. Precision designs are typically used when a report requires a precise layout, as is the case for invoices or bank checks. With SQL Server Report Designer, you can drag fields onto a report and put them where you want them. A precision design is free-form. Therefore, the format of a precision design can vary, depending on the layout that is required.

Datasets

A report dataset identifies the data that is displayed in a report. Dataset elements contain the information used to bind to a data source. After you define a dataset, you can reference the dataset when setting the *Dataset* property for a data region in the report design. If your report uses the predefined Microsoft Dynamics AX data source and a query that is defined in the AOT, be especially careful when updating the query in the AOT. For example, if you remove a field in the query and the field appears in the report, the report will display an empty column for the field. Whenever you make updates to a query, be sure to consider how those updates affect your reports. Updates to a query may also require updates to your reports.

The SSRS reporting framework supports five types of data connections:

- **Microsoft Dynamics AX queries** Access OLTP data by using a modeled collection of field data and table display methods. Microsoft Dynamics AX query objects defined in the AOT are used to define the data source, including the fields that are returned, record ranges, and relations to child data sources.

- **Report data providers (RDPs)** Access datasets derived from X++ business logic. An RDP data source is appropriate in cases where the following conditions are met:
  - You cannot query directly for the data that you want to render on a report.
  - The data to be processed and displayed is accessible from within Microsoft Dynamics AX.
- **SSAS OLAP queries** Access pre-aggregated views of Microsoft Dynamics AX business data. Microsoft Dynamics AX includes more than 10 predefined cubes. Use an OLAP data source to access pre-aggregated business data. For more information about cubes, see Chapter 10, "BI and analytics."
- **Transact-SQL (T-SQL) queries** Access data from external databases. With T-SQL–based connections, you can access data from external Microsoft SQL Server databases and use it within the report.
- **Internet services queries** Use data methods to access the data feeds provided by Internet service providers. For example, you can access industry-related data to compare the health of your business against the competition.

You have the option of relying on a single data source or you can combine data derived from multiple data sources to produce the report dataset. Identifying the best fit to satisfy your data access requirements greatly simplifies the design development and experience and improves the functionality and performance of the report.

SSRS extensions

The Microsoft Dynamics AX reporting framework takes advantage of several custom extensions supported by the SSRS platform to provide a fully integrated reporting experience that automatically adheres to security access rights and data formatting standards. This section provides some insights into how the reporting extensions function in the Microsoft Dynamics AX reporting framework.

Figure 9-8 illustrates the standard report execution sequence without Microsoft Dynamics AX custom extensions. (Figure 9-11, later in this chapter, illustrates the report execution sequence with Microsoft Dynamics AX custom extensions.)

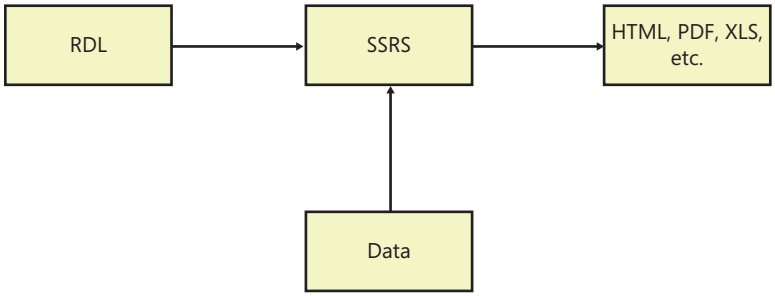


FIGURE 9-8 Standard report execution sequence.

Microsoft Dynamics AX extensions

The Microsoft Dynamics AX Report Definition Customization Extension (RDCE) is a reporting framework component introduced in Microsoft Dynamics AX 2012. It is internal to the reporting framework and is not directly accessible outside the framework. This component enables the reporting framework to provide run-time design alterations based on Microsoft Dynamics AX metadata and security policies. Dynamic transformation of RDL is needed for the following set of actions:

- Hide columns in reports if a user does not have access to those columns
- React to metadata changes in Microsoft Dynamics AX
- Use Microsoft Dynamics AX labels in reports
- Automatically flip designs for Microsoft Dynamics AX languages such as Arabic and Hebrew, which require right-to-left (RTL) layouts

A typical reason for hiding a column is security. In Microsoft Dynamics AX 2009, if a user didn't have access to a column, the data was not presented in the report but the column still appeared in the report (see Figure 9-9). This behavior is inconsistent with the legacy MorphX reporting framework and does not provide the ideal user experience.

AX2009 Behavior

User has access to **SALARY** column

NAME	SALARY
Akuma	$40,000
Ryu	$55,000
Ken	$69,000
Chen-Li	$75,000
Guile	$80,000

User does **NOT** have access to **SALARY** column

NAME	SALARY
Akuma	
Ryu	
Ken	
Chen-Li	
Guile	

FIGURE 9-9 Microsoft Dynamics AX 2009 user experience for SSRS reports.

In contrast, Microsoft Dynamics AX 2012 goes a step further and completely removes the column from the report design (see Figure 9-10). This is accomplished by means of the rendering extensions supplied by the reporting framework.

AX2012 Behavior

User has access to **SALARY** column

NAME	SALARY
Akuma	$40,000
Ryu	$55,000
Ken	$69,000
Chen-Li	$75,000
Guile	$80,000

User does **NOT** have access to **SALARY** column

NAME
Akuma
Ryu
Ken
Chen-Li
Guile

FIGURE 9-10 Microsoft Dynamics AX 2012 user experience for reports.

A number of features in Microsoft Dynamics AX 2012 require transformation of the RDL as part of the run-time processing. Conceptually, they are broken apart into separate RDL transformations; however, their implementation may be organized differently than the five discrete units shown in Table 9-1.

TABLE 9-1 RDL transformations.

Transformation	Auto design	Precision design	Transformation type
Microsoft Dynamics AX labels	Yes	Yes	Text content
RTL flipping	Yes	N/A	Layout
Auto size	Yes	N/A	Layout
Field groups	Yes	No	Layout
EDT column width	Yes	No	Layout
EDT numeric formatting	Yes	Yes	Text content
Task and role security	Yes	Yes	Layout
Configuration keys	Yes	Yes	Layout

Disable the rendering extensions

Many reporting scenarios do not rely on run-time design alterations based on user context information; instead, they require fast performance because of their scale. This is the case for most document-based reports, such as those listing customer and vendor invoices, purchase packing slips, and checks. Although the overhead of dynamic formatting of report designs is barely noticeable in an interactive session, it may become an issue in bulk operations where a large number of reports are requested as part of a batch operation. The Microsoft Dynamics AX reporting framework includes a control in the Report Deployment Settings form (Tools > Business Intelligence Tools > Report Deployment Settings) that disables the custom rendering extensions for specific report designs. This setting is highly recommended for any large-scale transactional reports that run in batch operations and don't require run-time design alterations.

If you select the Use Static Report Design check box in the Report Deployment Settings form, Microsoft Dynamics AX produces language-specific versions of the report design with labels and column sets fully resolved. This occurs the next time that the report is deployed to the report server. These reports are called *static RDL reports*. The Microsoft Dynamics AX reporting framework automatically uses the design that is appropriate given the context of the user running the report. However, no additional design alterations are performed when the report is invoked by the user. To make the report dynamic again, clear the Use Static Report Design check box or delete the entry in the Report Deployment Settings form.

Data processing extensions

Data processing extensions are used to query a data source and return a flattened row set. SSRS uses different extensions to interact with different types of data sources. A data source is simply the source of data for one or more reports. Data sources may be bound to Microsoft Dynamics AX or external databases, depending on the unique requirements of your reporting solution. Furthermore, you can display and interact with information from multiple data sources in a single report. Table 9-2 lists the types of data sources supported by the Microsoft Dynamics AX reporting framework.

TABLE 9-2 The types of data sources supported by the Microsoft Dynamics AX reporting framework.

Data source type	Data content
Microsoft Dynamics AX query	Access Microsoft Dynamics AX data by using predefined queries in the AOT.
Report Data Provider	Construct the data by using X++ business logic stored in specialized classes in the AOT.
OLAP	Access pre-aggregated views of your business data through SSAS.
SQL	Use T-SQL queries to access external databases.
Data methods	Connect to .NET service providers by using C# business logic.

Report execution sequence with Microsoft Dynamics AX custom extensions

The custom Microsoft Dynamics AX reporting extensions let you use a static design definition to produce dynamic reporting solutions that react to changes to Microsoft Dynamics AX metadata and user access rights.

Figure 9-11 illustrates the report execution sequence with the Microsoft Dynamics AX custom extensions in place.

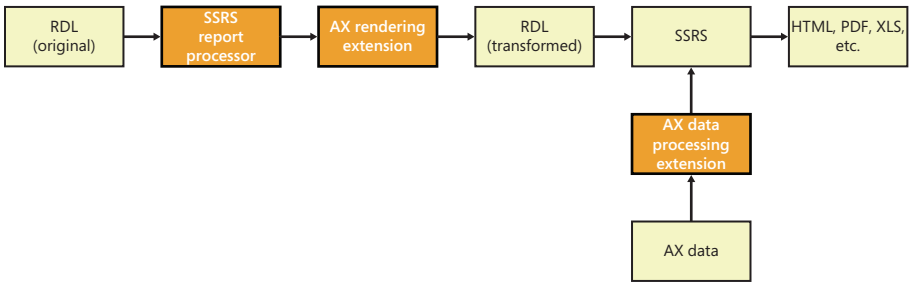


FIGURE 9-11 Report execution sequence with Microsoft Dynamics AX custom extensions.

Create charts for Enterprise Portal

This section discusses charting controls in Enterprise Portal and describes how they work. Charts provide a summary view of your data. With large datasets, charts often become obscured or unreadable. Missing or null data points, data types ill-suited to charts, and advanced applications such as combining charts with tables can all affect the readability of a chart. Before designing a chart, carefully prepare and understand your data and functional requirements so that you can design your charts quickly and efficiently.

Figure 9-12 shows some key elements that are used in a chart.

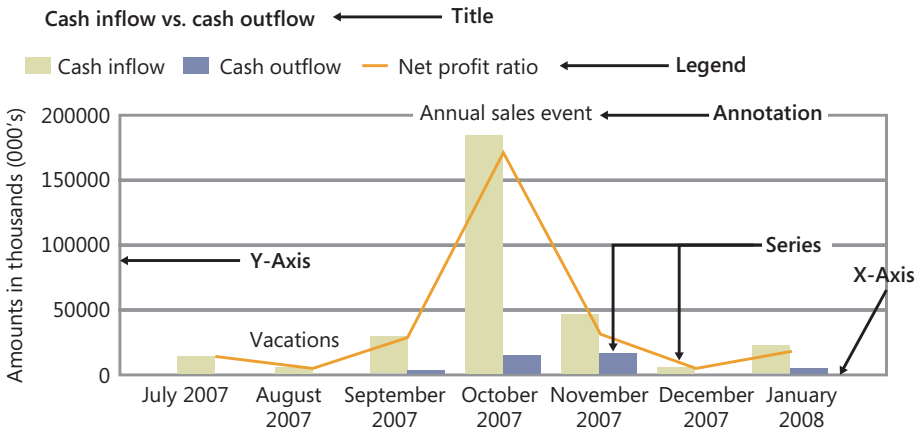


FIGURE 9-12 Chart elements.

Microsoft Dynamics AX chart development tools

The Microsoft Dynamics AX chart development tools simplify the development experience, making visualizing data easier through the EP Chart Control. This .NET chart control is installed during setup and the reporting framework handles all the work to gain access to it. You have access to all the functions and event handlers provided by the ASP.NET chart control to produce interactive and graphically rich data visualizations. The reporting framework extensions also provide the following features:

- Access to OLAP data sources by means of an MDX editor
- Access to OLTP data by means of a data source provider picker
- Built-in awareness of data partition integrations
- Microsoft Dynamics AX security access rights and policies
- Data formatting based on the Microsoft Dynamics AX Extended Data Type (EDT) definitions

Integration with Microsoft Dynamics AX

The definitions for chart controls are maintained in the AOT and in a distributed development environment, you can share them as XPO files. They are available in the list of Microsoft Dynamics AX user controls in Enterprise Portal as soon as you save them in the AOT. EP Chart Controls are maintained in the AOT along with other types of Dynamics AX user controls and can be deployed directly from the Development Workspace.

Figure 9-13 illustrates how chart controls are managed in the AOT.

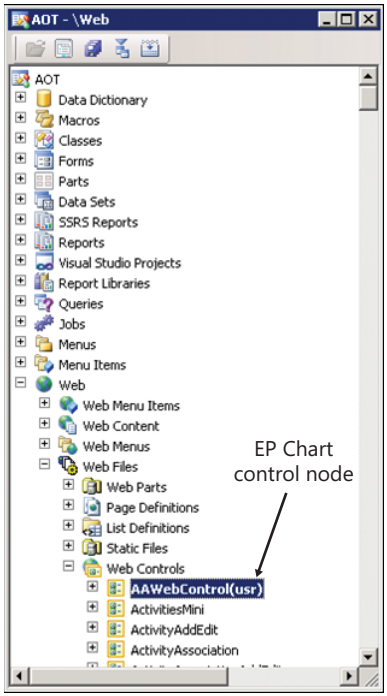


FIGURE 9-13 An EP Chart Control in the AOT.

Create an EP Chart Control

The Microsoft Dynamics AX development tools offer a new item template for Visual Studio called EP Chart Control (see Figure 9-14) to help get you started. This template contains all the required namespace definitions and the basic structure of a web control containing a single instance of the EP Chart Control.

The template adds an EP Chart Control to a project. The EP Chart Control has an empty chart area and series that are predefined. Pressing F5 starts the *http://localhost* script that you can use to debug your application. However, you will not see anything when you run your application because the series does not contain any data yet.

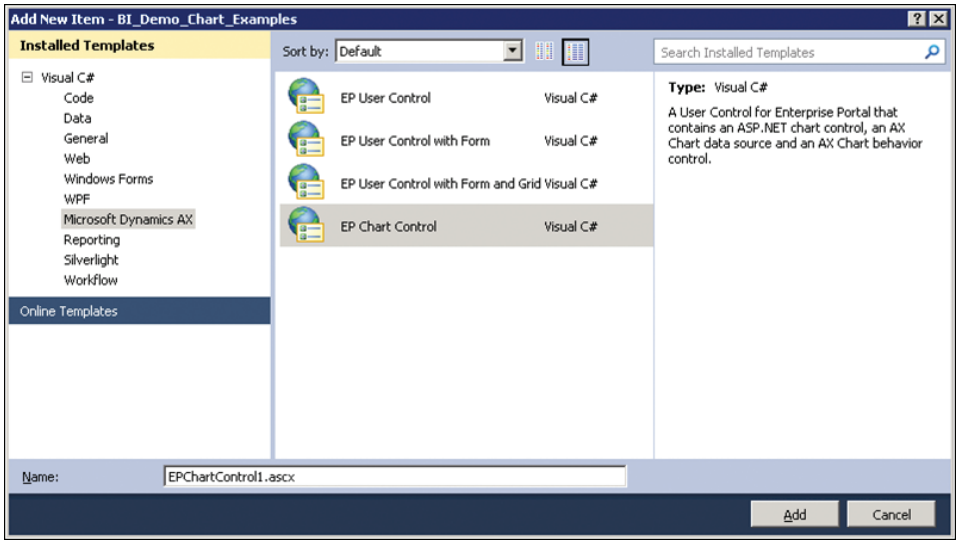


FIGURE 9-14 Adding an EP Chart Control.

Note If you encounter the error "Server Error in '/' Application," see the discussion of troubleshooting Microsoft SharePoint sandbox issues in the section "Troubleshoot the reporting framework," later in this chapter.

Chart control markup elements

The standard ASP.NET chart control is represented in ASPX markup code by the *<asp:Chart>* element. In Visual Studio, the markup for the EP Chart Control includes two additional controls: *<dynamics:AxChartDatasource>* and *<dynamics:AxChartBehavior>*. You use these three elements in concert to define the appearance and functions of the EP Chart Control and manage the connection information to the underlying data source.

- ***<asp:Chart>*** Maps to the ASP.NET chart control that is available as a download for Microsoft.NET Framework 3.5 and included with .NET Framework 4.0. This element is used to define the general structure of the control. For more information, see "Technical Reference: Chart Controls," at *http://msdn.microsoft.com/en-us/library/dd456726*.
- ***<dynamics:AxChartDatasource>*** Contains the data source connection type, along with the query that is used to access the data. This element is also where access parameters are defined to construct the query, if required. You can access the Visual Studio tools for defining data connections to Microsoft Dynamics AX data through the web control designer for this element.
- ***<dynamics:AxChartBehavior>*** Supplies default formatting for chart controls. You can use this element to define custom color palettes for your chart solutions or to disable the reporting framework's default formatting engine. You also use this element to define the structure of static and dynamic datasets by means of element properties.

Figure 9-15 contains a screenshot of the EP Chart Control in Design mode.

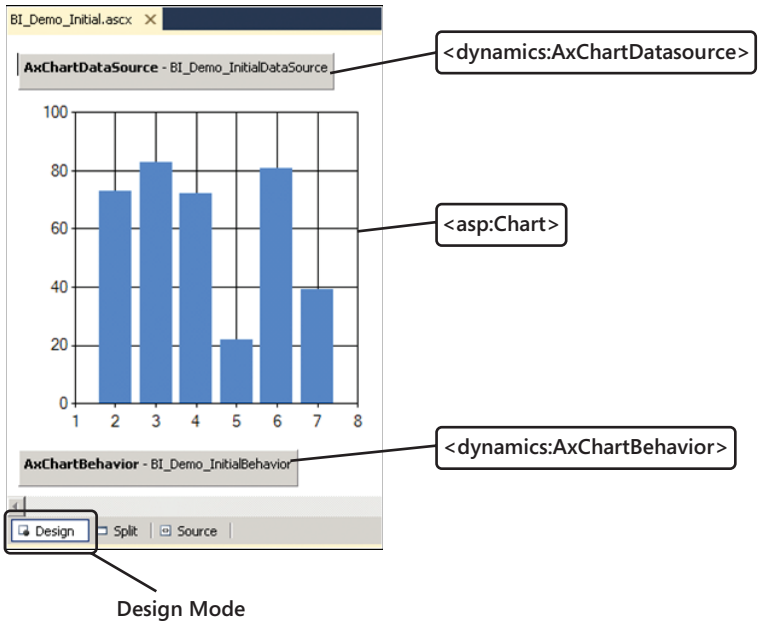


FIGURE 9-15 EP Chart Control in Design mode.

Bind the chart control to the dataset

The first task in creating a chart control is to bind the chart control to the dataset. The Visual Studio development environment has been extended to include tools to simplify the process of binding to Microsoft Dynamics AX data. Two categories of data sources are derived from Microsoft Dynamics AX:

- **OLTP data sources** Data that is managed in the AOS exposed through the AX Query Service interface. Declarative connections to Report Data Providers are made available using data source picker control.
- **OLAP data sources** Aggregate data managed by the Microsoft Dynamics AX analytics framework. The Visual Studio extensions offer an MDX editor to help you create queries to access data stored in an SSAS database.

Data series

This section summarizes basic data binding strategies that you can use when visualizing data with charts. The EP Chart Control supports three basic data binding scenarios: single series datasets, multiseries datasets, and dynamic series datasets.

Single series datasets

In a single series dataset, the source data for the chart can be described by using only two columns, as shown in Figure 9-16. A single series dataset is most commonly used when figures have a single pivot; for example, trending over time or distribution across segments. Stock performance over time is an example where only two columns are required. Pie charts, bar charts, column charts, and funnel charts are commonly used to visualize single series datasets.

DATASET	
MONTH	MSFT
January 2005	200
February 2005	300
March 2005	400
May 2005	105
July 2005	107
January 2006	500
June 2007	258

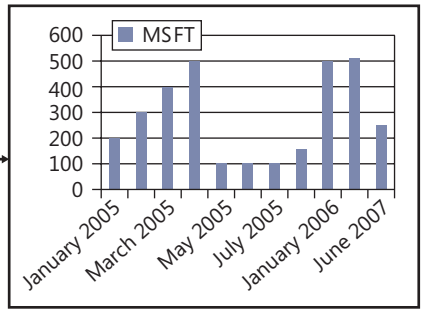


FIGURE 9-16 Chart from a single series dataset.

Multiseries datasets

Multiseries datasets require at least three columns, as shown in Figure 9-17. Individual series elements are bound to a set of columns defined by the dataset. With this type of dataset, you can compare figures by two pivots in that you can additionally gain relative analysis by comparing against related data. Bar charts and column charts along with many others are suitable for analyzing multiseries datasets. However, pie charts and funnel charts are not appropriate for visualizing multiseries datasets.

DATASET		
MONTH	MSFT	GOOG
January 2005	200	400
February 2005	300	400
March 2005	400	350
May 2005	105	320
July 2005	107	250
January 2006	500	450
June 2007	258	350

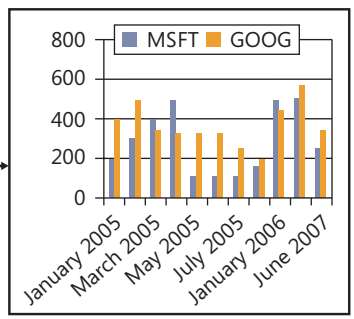


FIGURE 9-17 Chart from a multiseries dataset.

Dynamic series datasets

Often, the number of series is defined within the dataset itself. These datasets are referred to as *dynamic series datasets* and can be described using three or more columns, as shown in Figure 9-18. The biggest differentiator between a dynamic series dataset and a multiseries dataset is the inclusion of a column that identifies the unique series. Dynamic series datasets are appropriate in cases where the number of series is determined by the user or by attributes that are related to the data source. Dynamic series datasets can be viewed by using the same types of charts as multiseries datasets.

DATASET		
MONTH	SERIES	VALUE
January 2005	MSFT	400
January 2005	GOOG	400
March 2005	MSFT	405
March 2005	GOOG	505
March 2005	AAPL	110
March 2005	YHOO	116
January 2006	GOOG	117
January 2006	GOOG	168
April 2006	AAPL	523
June 2007	YHOO	136
June 2007	GOOG	268

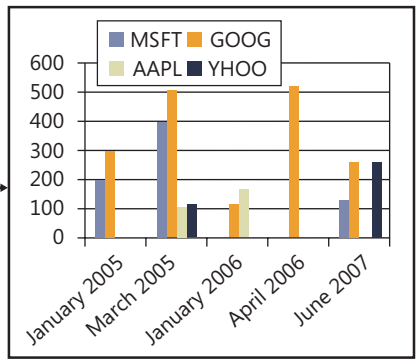


FIGURE 9-18 Chart from a dynamic series dataset.

For more information about how to choose the right type of chart for your data, see "Chart Types (Report Builder and SSRS)," at *http://msdn.microsoft.com/en-us/library/dd220461(v=sql.110).aspx*.

Add interactive functions to a chart

Each series in a chart consists of a set of data points, which, for most chart types, is made up of two key attributes: *X* and *Y* values. Collectively, the control uses these data points to render the data series in a method that is consistent with the type of chart you select. In addition to *X* and *Y* values, data points can contain additional information, including drill-through URLs, tooltip text, and data point labels. As you would expect, when a data point contains a definition for a drill-through URL, the data point becomes clickable in the chart image. When the user clicks the data point, he or she is taken to the specified URL. Defining tooltip text for a data point automatically produces a tooltip containing the text when the user hovers over the data point in the chart. You can extend the original dataset by using a post-processing event handler to include additional data-point information that drives the interactive experience provided by the chart.

Follow these basic steps to expand the chart dataset to include interactive functions:

1. Access the data that you want to appear in the chart.
2. Add post-processing code that expands the schema of the underlying data table.
3. Format the data columns based on their intended use.
4. Bind newly created columns to the chart properties that control the interactive functions of the control when it is rendered for the user.

Figure 9-19 illustrates the sequence for expanding the dataset to add columns that are formatted for interactive use.

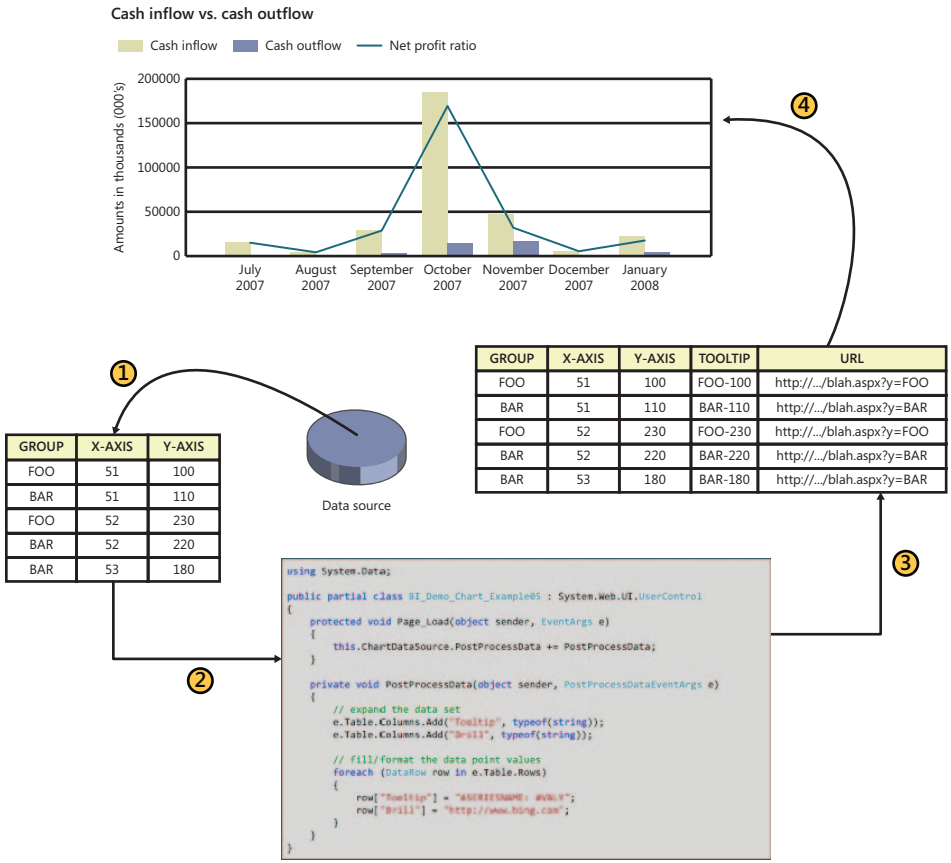


FIGURE 9-19 Expanding a dataset to create an interactive chart.

Override the default chart format

The reporting framework applies some default formatting to the most common types of controls. Default formatting is applied for three main reasons:

- To promote consistency among charts that are displayed in Enterprise Portal
- To simplify the development experience
- To ensure that charts are visually compelling to users

At times, however, you may want to apply formatting that differs from defaults. You can override the default design formatting by using control event handlers.

It is recommended that you customize the EP Chart Control in response to the *PreRender* event. This is where you define code executed at run time to manage the format of the EP Chart Control. Customizations can include dynamic color palettes, custom label positioning, and text formatting.

Figure 9-20 demonstrates the basic steps in adding code to override the default formatting.

```
using System.Data;

public partial class BI_Demo_Chart_Example05 : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.ChartBehavior.PreRender += new EventHandler(PreRender);
    }

    protected void PreRender(object sender, EventArgs e)
    {
        //Format the pie label style
        this.Chart.Series["Series1"]["PieLabelStyle"] = "Inside";

        //Format the data points
        foreach (DataPoint point in this.Chart.Series["Series1"].Points)
        {
            point.LegendText = point.Axislabel;
            point.AxisLabel = "#PERCENT";
            point.ToolTip = "#LEGENDTEXT" + ": + "#VALY{C}";
            point.LabelForeColor = ChartColors.Black;
        }
    }
}
```

FIGURE 9-20 Overriding default chart formatting.

For more information about events, see "ASP.Net Page Life Cycle Overview," at *http://msdn.microsoft.com/en-us/library/ms178472.aspx*.

Troubleshoot the reporting framework

This section contains some of the most common reporting framework issues and possible solutions. You can find related information posted on the Microsoft Dynamics AX Product Forum at *https://community.dynamics.com/product/ax/f/33.aspx*.

The report server cannot be validated

If you cannot validate the report server, do the following:

- Click the Create button in the Reporting Servers form, which is located at Tools > Business Intelligence Tools > Reporting Servers, and make sure that a report folder and data source have been created on the report server. Click the Validate button.
- Ensure that firewall settings are configured appropriately on the computer that is running the report server.
- Ensure that both the report manager and report server URLs are correct.
- Ensure that the Microsoft Dynamics AX user has permissions on the computer that is running the report server.

A report cannot be generated

If you are connecting to the Microsoft Dynamics AX SQL Server database and the system will not generate a report, do the following:

- Ensure that the report server account configured in the report data source on the report server has read permissions on the Microsoft Dynamics AX SQL Server database.
- Ensure that firewall settings are configured appropriately on the computer on which the database is installed.

If you are connecting to an external or custom data source, make sure that the user name and password provided for the report server account in the data source on the report server are correct.

A chart cannot be debugged because of SharePoint sandbox issues

If you cannot debug a chart because of problems with the SharePoint sandbox, do the following:

- Add a reference to the *Microsoft.SharePoint.dll* assembly to the project.
- Establish the default web control to run in debug mode.
- Edit the file named *Default.aspx* in the EP Chart project.
- Add the *ManagedContentItem* property to the <*dynamics:AxUserControlWebPart*> element and set the value to the name of the web control.

CHAPTER 10

BI and analytics

In this chapter

Introduction 299
Components of the Microsoft Dynamics AX 2012 BI solution 299
Implementing the prebuilt BI solution 301
Customizing the prebuilt BI solution 309
Creating cubes 323
Displaying analytic content in Role Centers 333

Introduction

Business Intelligence (BI) technology helps users of computer-based applications understand hidden trends and exceptions within data. Nowadays, it's difficult to find a developer who is unaware of BI, so this chapter assumes that you are familiar with BI concepts.

Microsoft Dynamics AX 2012 includes a comprehensive prebuilt BI solution, which is designed to meet many of the BI needs of your users. This means that instead of having to build a BI solution from the ground up, you may be able to use the prebuilt solution and tweak it to meet any remaining requirements. With this proposition in mind, this chapter walks you through the life cycle of the Microsoft Dynamics AX 2012 analytic components—from implementation through customization and extension. When necessary, this chapter points you to relevant resources on the Internet.

The Microsoft Dynamics AX BI solution is built on top of the Microsoft BI framework. If your organization uses the Microsoft BI infrastructure, you can use the Microsoft BI tools and technologies to extend the power of the Microsoft Dynamics AX BI solution.

Components of the Microsoft Dynamics AX 2012 BI solution

Figure 10-1 shows a simplified architecture diagram of the BI solution that is included with Microsoft Dynamics AX 2012. In the figure, the Microsoft Dynamics AX 2012 logical architecture has been simplified to highlight only the components that are relevant to the BI solution.

The solution is divided into three tiers:

- **Data tier** Contains sources of data, such as the Microsoft Dynamics AX 2012 operational database, often referred to as the *online transaction processing (OLTP) database*.

- **Integration tier** Contains the Application Object Server (AOS), programming interfaces, and staged data, such as Microsoft Dynamics AX 2012 cubes, that serve as the database for analytical reporting. (This tier is called the middle tier in Chapter 1, "Architectural overview." It is called the integration tier in this chapter because that is how it is commonly known in BI solutions.)
- **Presentation tier** Contains tools and user interface elements that users can use to interact with data.

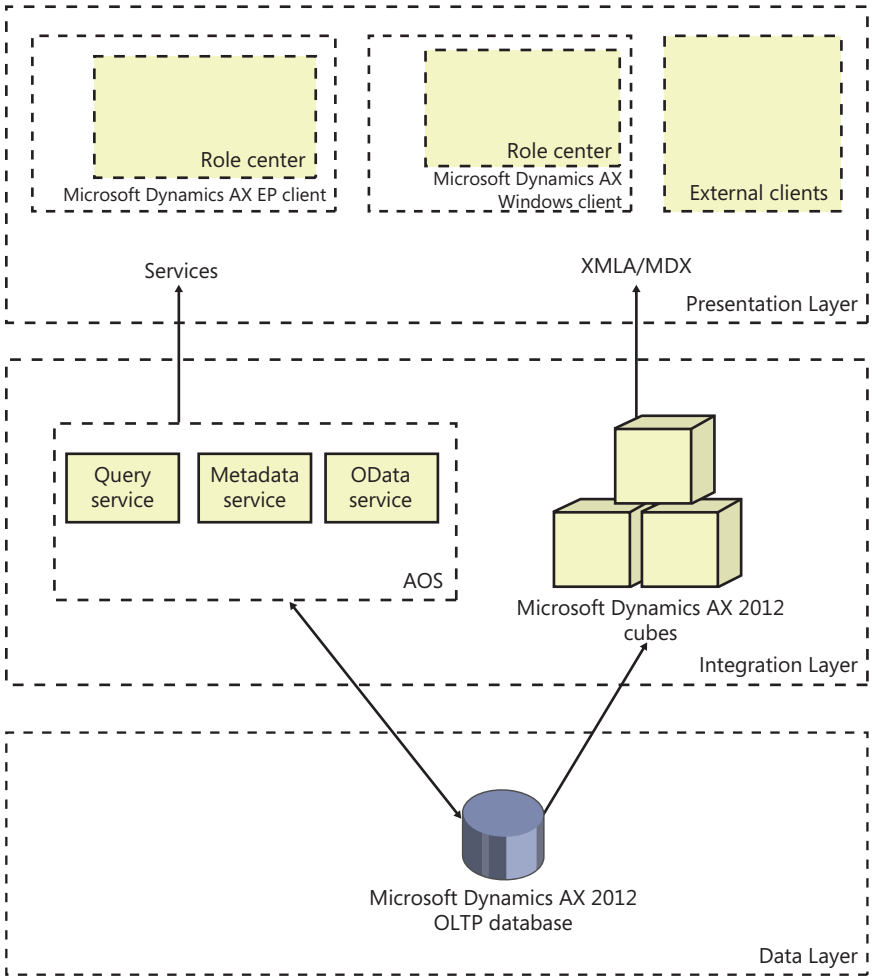


FIGURE 10-1 Microsoft Dynamics AX BI architecture.

For details about the three tiers and a more detailed diagram, see Chapter 1.

Implementing the prebuilt BI solution

Traditionally, BI solutions are implemented during the second or third phase of an Enterprise Resource Planning (ERP) implementation project. Needless to say, project fatigue sets in (and the budget gets exhausted), and subsequent phases are postponed or delayed. BI implementation is complex and involves the integration of many components. Also, the skill set required to implement a BI solution is distinctly different from the skill set required to implement an ERP system. Often, implementation of the BI solution involves engaging a different partner or consultants. All of these factors contribute to postponing the BI implementation.

Microsoft Dynamics AX 2012 simplifies the implementation of a BI solution, so that all Microsoft Dynamics AX 2012 partners and customers (regardless of whether they have access to BI specialists) can implement the prebuilt BI solution when they implement the ERP functionality.

In Microsoft Dynamics AX 2012, the default SQL Server Analysis Services (SSAS) project is a first-class citizen of the Application Object Tree (AOT), as are other SSAS projects that you create in the AOT. This means that SSAS projects derive all of the benefits of being residents of AOT.

- SSAS projects respect the layering concept. This means that an independent software vendor (ISV) or partner can distribute a customized version of an SSAS project that adds additional analytic components to the solution that is included in the SYS layer.
- You can import and export SSAS projects to and from different environments as part of a model (by using models or .xpo files).
- SSAS projects respect the version control capabilities offered by AOT-based artifacts.

When you deploy a project by using the SQL Server Analysis Services Project Wizard, which is new in Microsoft Dynamics AX 2012, the wizard selects the project in the highest layer for deployment. If you examine the *Visual Studio Projects* node in the AOT, will see the default SSAS project that is included with Microsoft Dynamics AX 2012, as shown in Figure 10-2. If you have any customizations at higher levels, they are also displayed.

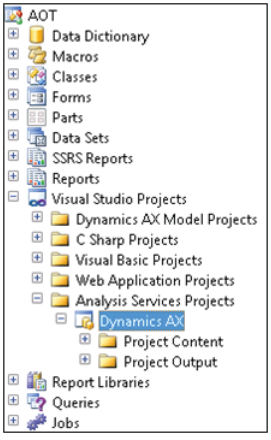


FIGURE 10-2 SSAS projects in the AOT.

Implementing the prebuilt BI solution consists of the following steps:

1. Implement the prerequisites.
2. Configure an SSAS server.
3. Deploy the cubes.
4. Process the cubes.
5. Provision users so that they can access the analytic data.

The following sections describe each step in further detail.

Implement the prerequisites

Before you implement the analytic components in the prebuilt BI solution, the following Microsoft Dynamics AX core components should be in place:

- At least one AOS instance must be implemented.
- The Microsoft Dynamics AX Windows client must be implemented, and the initialization checklist must be completed.
- The Enterprise Portal web client must be configured.

If you are implementing the analytic components on a development or test instance, you might not implement a scale-out architecture. However, if you are implementing these components in a production system, you may want to implement a redundancy or load balancing infrastructure. You need to configure the clustering or Network Load Balancing (NLB) solution before you implement the analytic components.

Configure an SSAS server

This step configures a given SSAS server for the Microsoft Dynamics AX 2012 analytic components. To do so, run the Configure Analysis Extensions step in the Microsoft Dynamics AX Setup wizard on the SSAS server that hosts Microsoft Dynamics AX 2012 cubes.

Running the configuration step should take you a few minutes. This function does the following:

- Ensures that the SSAS server has all of the necessary prerequisites to host Microsoft Dynamics AX 2012 cubes.
- Adds the Business Connector (BC) proxy user as an administrator of the SSAS server. This step is required to enable AXADOMD data extensions to operate without the use of Kerberos constrained delegation.
- Allows you to add a read-only user account to the Microsoft Dynamics AX 2012 database for processing cubes (you should specify a domain account whose password does not expire).

Deploy cubes

When you deploy cubes, Microsoft Dynamics AX generates and processes an OLAP database by using the metadata definition contained within the Analysis Services project that is included with Microsoft Dynamics AX 2012. The result is an OLAP database that contains Microsoft Dynamics AX cubes that are referenced by analytic reports and Role Centers.

In a Microsoft Dynamics AX 2012 R2 environment where there is only a single partition, the deployment step generates a single OLAP database that sources data from the Microsoft Dynamics AX OLTP database. In a multiple-partition environment, the deployment step generates multiple OLAP databases that correspond to each partition. Figure 10-3 shows the deployment process both in a single-partition and multiple-partition environment. For more information about partitions, see Chapter 17, "The database layer."

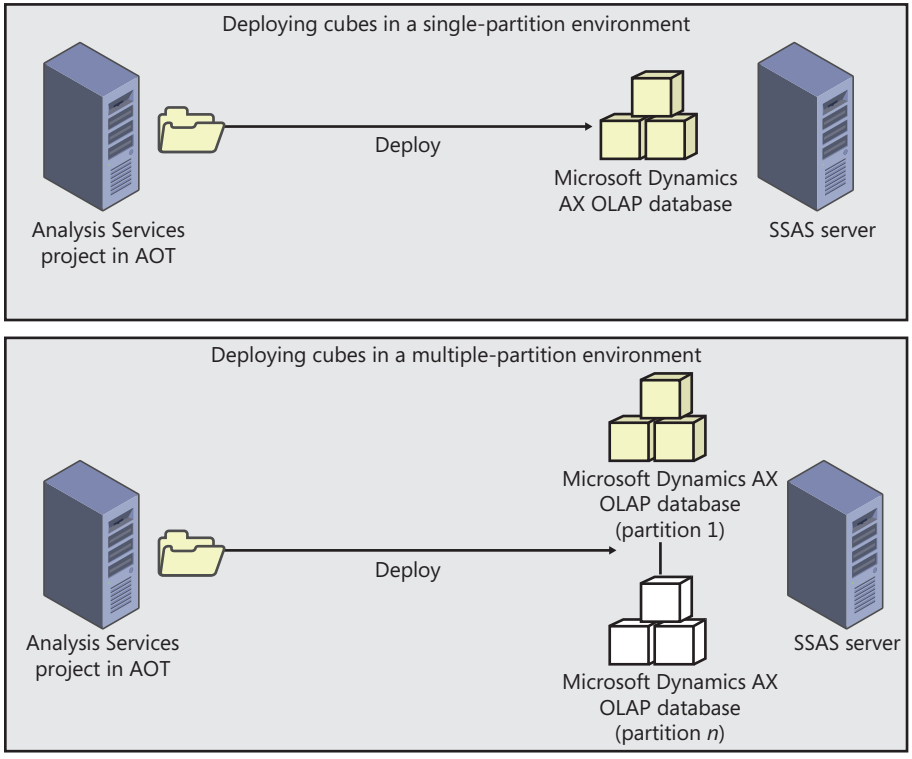


FIGURE 10-3 Deploying cubes in single-partition and multiple-partition environments.

You use the SQL Server Analysis Services Project Wizard in the Microsoft Dynamics AX 2012 client to deploy, process, and in some instances, update cubes. To deploy the cubes, you must have the right to deploy projects to the SSAS server. If you are also processing the cubes, you must have the right to read the Microsoft Dynamics AX 2012 OLTP database.

To start the SQL Server Analysis Services Project Wizard and deploy cubes, do the following:

1. In the Development Workspace, on the Tools menu, click Business Intelligence (BI) Tools > SQL Server Analysis Services Project Wizard.

2. On the Welcome page, click Next, and then select the Deploy option on the next page, as shown in Figure 10-4.

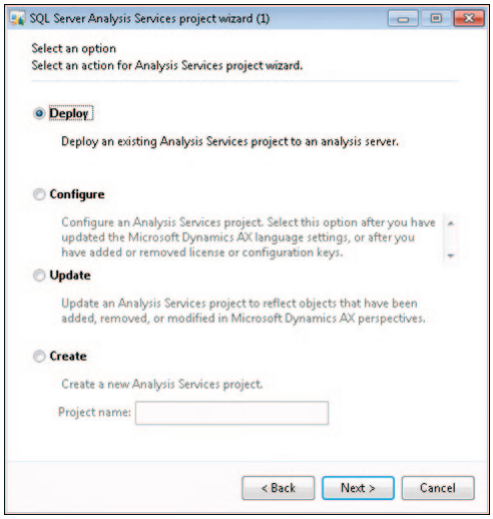


FIGURE 10-4 The Deploy option in the SQL Server Analysis Services Project Wizard.

3. On the next page, you select an SSAS project to deploy—in this case the Dynamics AX project. You can select a project in the AOT, as shown in Figure 10-5, or you can select a project that is saved on a disk.

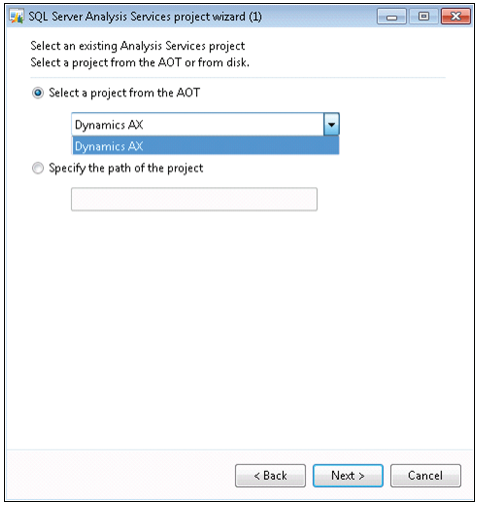


FIGURE 10-5 Selecting an SSAS project.

4. Next, you specify the SSAS server to deploy the project to, the SSAS database you want to use, and whether you want the project to be processed after deployment (see Figure 10-6). By default, the wizard uses the SSAS server that you configured earlier, but you can select any server to deploy the project to.

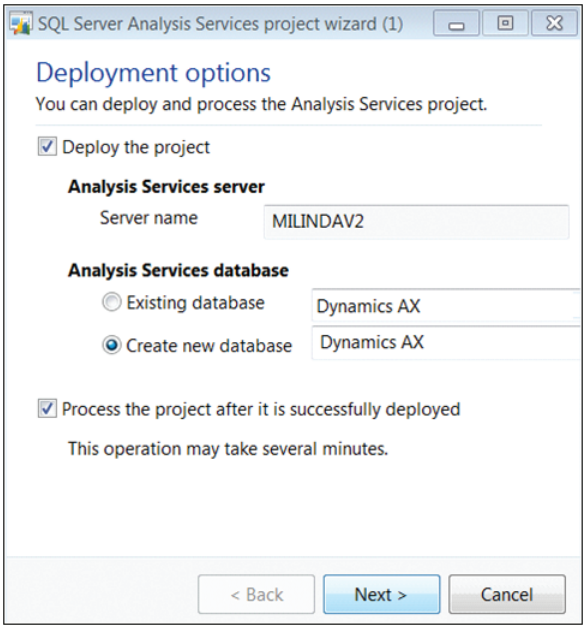


FIGURE 10-6 Deploy an SSAS project to a server in Microsoft Dynamics AX 2012.

Note In Microsoft Dynamics AX 2012, you can use any name for the OLAP database. In Microsoft Dynamics AX 2009, you couldn't change the default name of the database, and this prevented a system administrator from using the same SSAS server to host multiple OLAP databases. However, if you do change the default name of the OLAP database, you need to configure the report server so that it reports source data from the corresponding OLAP database. For information about how to configure the OLAP database referenced by SQL Server Reporting Services (SSRS) reports, see "Configure Analysis Services by running Setup" at *http://msdn.microsoft.com/en-us/library/gg751377.aspx*.

Deploy cubes in an environment with multiple partitions

As mentioned earlier, in a Microsoft Dynamics AX 2012 R2 environment with multiple partitions, the SQL Server Analysis Services Project Wizard generates an OLAP database for each partition. You can use the wizard to select the partitions for which OLAP databases are created, as shown in Figure 10-7.

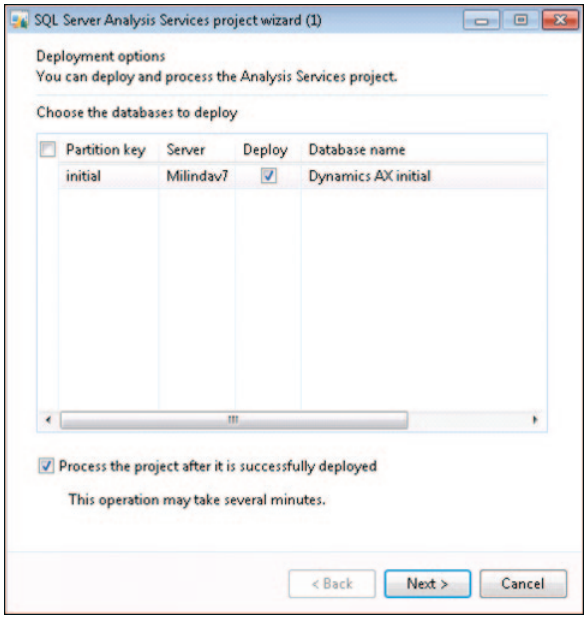


FIGURE 10-7 Selecting a partition in Microsoft Dynamics AX 2012 R2.

In this case, the SQL Server Analysis Services Project Wizard deploys the SSAS project to multiple OLAP databases. In each database, <*partitionkey*> is added as a suffix to the name of the OLAP database.

Also, within each OLAP database, the data source view (DSV) is modified so that a partition filter is applied to all queries. Figure 10-8 shows the architecture of an environment with multiple partitions.

In all cases, the SSAS project in the AOT is partition-unaware, whereas the OLAP databases that are deployed are partition-specific. The SQL Server Analysis Services Project Wizard handles the step of making sure that each OLAP database is wired to read data only from the corresponding partition in Microsoft Dynamics AX. This is a departure from the behavior of Microsoft Dynamics AX 2012. You need to be aware of the following implications:

- If you deploy Microsoft Dynamics AX SSAS projects by using Analysis Services tools, such as the Deployment Wizard or Business Intelligence Development Studio, the resulting OLAP database is not partition-aware. In other words, cubes will aggregate data across partitions.
- If you want to extend an SSAS project, always check out and modify the project in the AOT. Do not customize a project associated with a specific partition by importing the project directly in Business Intelligence Development Studio. The Deploy function in the wizard will overwrite any partition-specific customizations that you have made directly on the server.
- If you add custom query definitions in the DSV, the wizard adds *where* clauses to each *select* statement that restrict rows from other partitions.

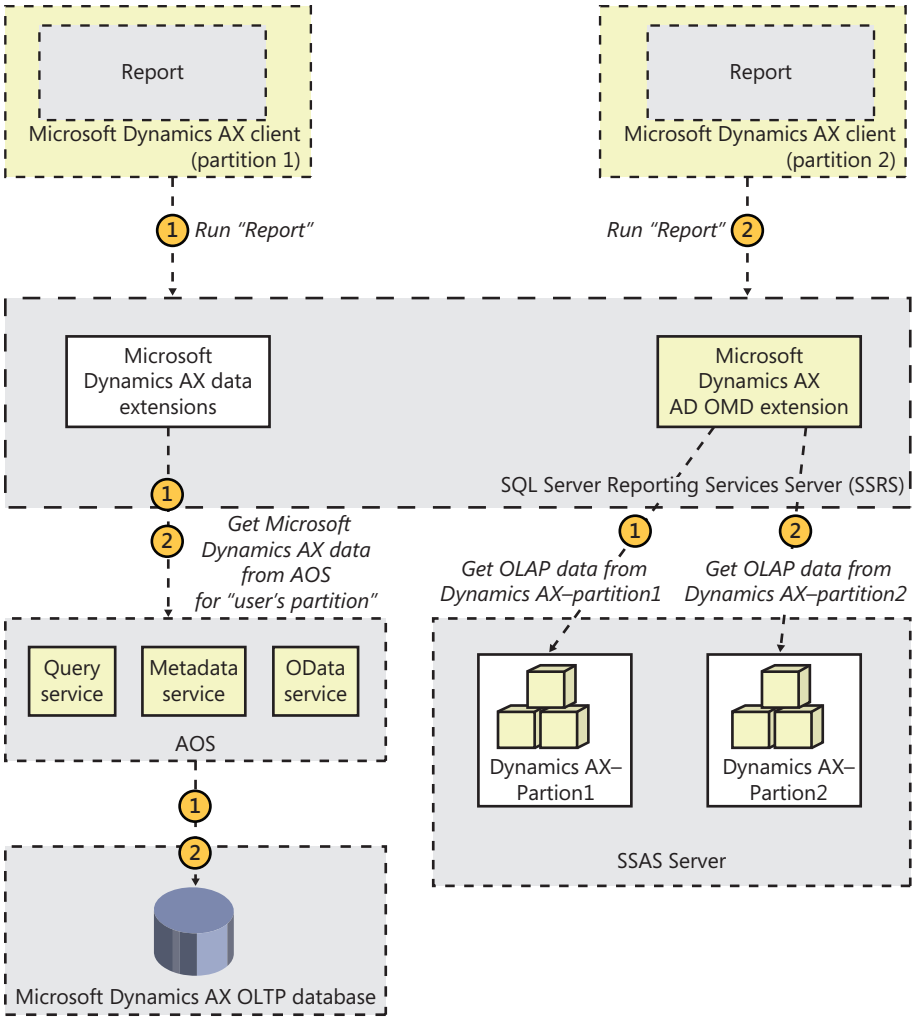


FIGURE 10-8 Architecture of an environment with multiple partitions.

Process cubes

The SQL Server Analysis Services Project Wizard lets you process deployed cubes directly. However, before processing, the wizard also runs through several prerequisite checks to ensure that cube processing will not fail later. If you are using demo data, you can ignore these preprocessing warnings and have the wizard process the cubes.

While the project is being processed, the wizard displays a progress page. When processing is complete, click Next, and the wizard will show the completion screen.

Provision users in Microsoft Dynamics AX

After you deploy and process Microsoft Dynamics AX cubes, you must grant users permissions to access them. Provisioning users involves two activities:

- Associate an appropriate user profile with each Microsoft Dynamics AX user.
- Provide Microsoft Dynamics AX users with access to the OLAP database.

Associate a user with a profile

The concept of a user profile was introduced in Microsoft Dynamics AX 2009. A user profile determines the Role Center that is displayed when a user starts the Microsoft Dynamics AX client. A user can be associated with only one profile.

If you do not associate a user profile with a user in Microsoft Dynamics AX, the default Role Center is displayed when the user displays the Home area page in the Microsoft Dynamics AX client. To associate a profile with a given user, click System Administration > Common > Users > User Profiles (see Figure 10-9). You can associate either one user at a time or multiple users with a given profile by using this form.

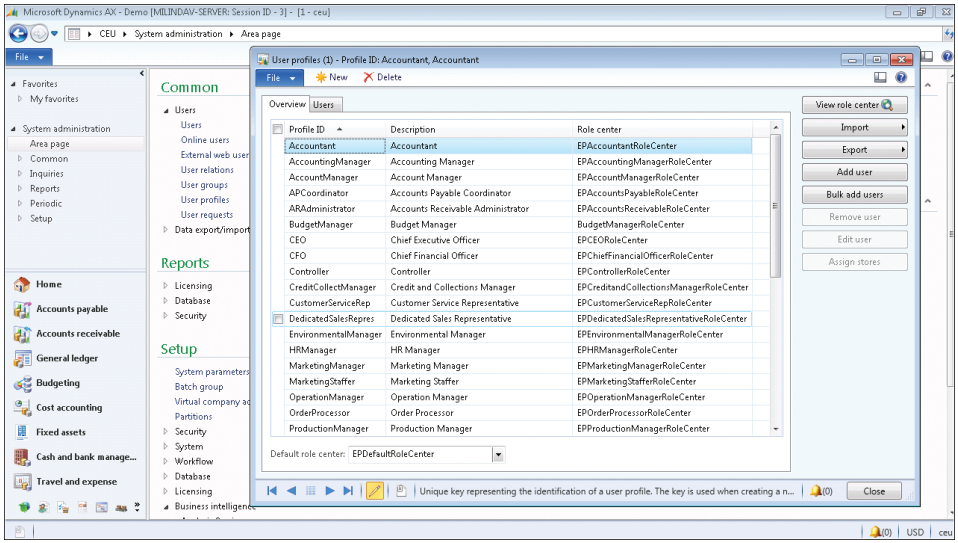


FIGURE 10-9 Associating a user with a profile.

You can also associate a user with a profile in the Users form (System Administration > Common > Users > Users). Changes to a user profile take effect the next time the user starts the Microsoft Dynamics AX client.

Provide access to the OLAP database

Unless you provide your users with access to the OLAP database, they cannot open reports and display key performance indicators (KPIs) drawn from cubes in their respective Role Centers. Security permissions defined in Microsoft Dynamics AX 2012 are not automatically applied to OLAP databases. You must grant access to OLAP databases manually by using SQL Server management tools, such as SQL Server Management Studio. For step-by-step instructions, see "Grant users access to cubes" at *http://msdn.microsoft.com/en-us/library/aa570082.aspx*.

Customizing the prebuilt BI solution

As you have seen in the previous section, it's relatively easy to implement the prebuilt BI solution in Microsoft Dynamics AX 2012. But regardless of how good the prebuilt BI solution is, you may want to change the functionality to suit your needs.

These changes can be divided into three broad categories:

- **Configuration** Although the prebuilt BI solution is designed to cover all of the functionality in Microsoft Dynamics AX 2012, you may have implemented only certain modules. Even within those modules, you may have chosen to disable certain functionality. In Microsoft Dynamics AX, license codes and configuration keys govern the availability of modules and functionality, respectively. (For more information, see Chapter 11, "Security, licensing, and configuration.") Configuration keys correspond to functionality within modules. They can be enabled or disabled.

  If you do not activate certain license codes or if you disable certain configuration keys, the Microsoft Dynamics AX user interface configures itself by removing content that is associated with those elements. In this case, you may need to remove the corresponding analytic content. (However, because the prebuilt BI solution draws data from across Microsoft Dynamics AX, this content will not be hydrated with data in any case.) You can use the SQL Server Analysis Services Project Wizard to remove the corresponding content from the prebuilt cubes, so that you do not have to remove the irrelevant content manually yourself.

- **Customization** You might want to add additional calendars and financial dimensions, and also new attributes and measures, to the prebuilt cubes. The SQL Server Analysis Services Project Wizard lets you perform the most frequent customizations with a step-by-step approach, without requiring BI development skills.

- **Extension** At some point, you may want to develop extensions to prebuilt cubes by using the SQL Server BI development tools. Table 10-1 lists categories of customizations, summarizes the types of changes that you can make, and lists the skill level, time, and tools required to make those types of changes.

TABLE 10-1 Types of customizations.

	Configuration	Customization	Extension
Nature of change	Apply the Microsoft Dynamics AX configuration to cubes; add or remove languages	Add calendars or financial dimensions; add or remove measures and dimensions	Any
Skills	Knowledge of Microsoft Dynamics AX concepts	Ability to define Microsoft Dynamics AX metadata	BI development skills
Tools	SQL Server Analysis Services Project Wizard	AOT; SQL Server Analysis Services Project Wizard	Business Intelligence Development Studio
Time required	Low	Medium	High

The following sections describe the processes for customizing the Microsoft Dynamics AX 2012 prebuilt BI solution.

Configure analytic content

As previously explained, you can configure the predefined analytic content to reflect configuration changes in Microsoft Dynamics AX in a matter of minutes by using the SQL Server Analysis Services Project Wizard. In Microsoft Dynamics AX 2009, this process had to be performed manually. This process required BI development skills and a day or two of spare time. Microsoft Dynamics AX 2012 dramatically simplifies this process by introducing the following three improvements:

- **Static schema** Historically, Microsoft Dynamics AX has had a schema whose shape changed depending on licenses and configuration keys. That is, when a configuration key was turned off, the database synchronization process dropped tables and data that were deemed invalid. This caused prebuilt cubes (that rely on a static schema in the underlying database) to break at processing time. Unlike its predecessor, Microsoft Dynamics AX 2012 has a static schema. So, when configuration keys are disabled, the database schema no longer changes. This means that prebuilt cubes can continue to be processed without generating errors. (They will, for example, contain empty measures, because the corresponding tables have no data).
- **Improved modeling capabilities in the AOT** The Microsoft Dynamics AX 2009 OLAP framework did not allow advanced modeling of constructs in the AOT. As a result, developers had to implement any functionality that was lacking directly in an SSAS project. In Microsoft Dynamics AX 2012, a larger portion of analytic content is modeled in the AOT. Therefore, configuring the content can be done much more easily by the framework.
- **Wizard-driven user interface** The six different forms that were necessary in Microsoft Dynamics AX 2009 have been replaced by a single step-by-step wizard that guides you through various activities.

To configure the prebuilt BI project, you must have developer privileges in Microsoft Dynamics AX. This step modifies the project so that irrelevant measures, dimensions, and entire cubes are removed after the process is completed. The modified project will be saved in the AOT in your own layer.

To configure the project, start the SQL Server Analysis Services Project Wizard, and then select the Configure option. You then need to select the project to configure. Select the Dynamics AX project to configure the prebuilt project, and step through the wizard. For step-by-step instructions, see the "How to: Configure an Existing SQL Server Analysis Services Project" at *http://msdn.microsoft.com/en-us/library/gg724140.aspx*.

If you also deploy and process the project, you should notice the following changes:

- Cube content (such as measures and dimension attributes that source data from tables that are affected by disabled configuration keys) is deleted from the project. You may see that entire cubes have been removed if the corresponding content has become invalid.
- KPIs and calculated measures have been removed in cubes that depend on disabled measures and dimension attributes.
- OLAP reports in Role Centers that source data from cubes that have been removed no longer appear on the Role Center page. If a user intentionally adds such a report to the Role Center, the report displays a warning message and will execute.
- KPIs and measures that were removed no longer appear in the Business Overview web part.

Customize cubes

When you start the SQL Server Analysis Services Project Wizard, the third option after Deploy and Configure is Update. This option lets you customize the project.

Figure 10-10 shows the process for updating a cube. The following sections walk through each step in detail.

Choose the project to update

The first step is selecting the project to modify. You can select an SSAS project in the AOT or a project maintained on disk. The wizard performs basic validation of the selected project before you can proceed. The update process is designed to ensure that you end up with a project that you can deploy and process without any errors. If the selected project does not build (the most basic measure of validity), the wizard will not let you proceed to the next step.

Select metadata

Next, you select the Microsoft Dynamics AX metadata that you want to include or exclude, as shown in Figure 10-11. The metadata that is defined in the *Perspectives* node in the AOT is the source of metadata for the prebuilt BI solution. By including or excluding metadata definitions, you can include (or exclude) measures, dimensions, and even cubes.

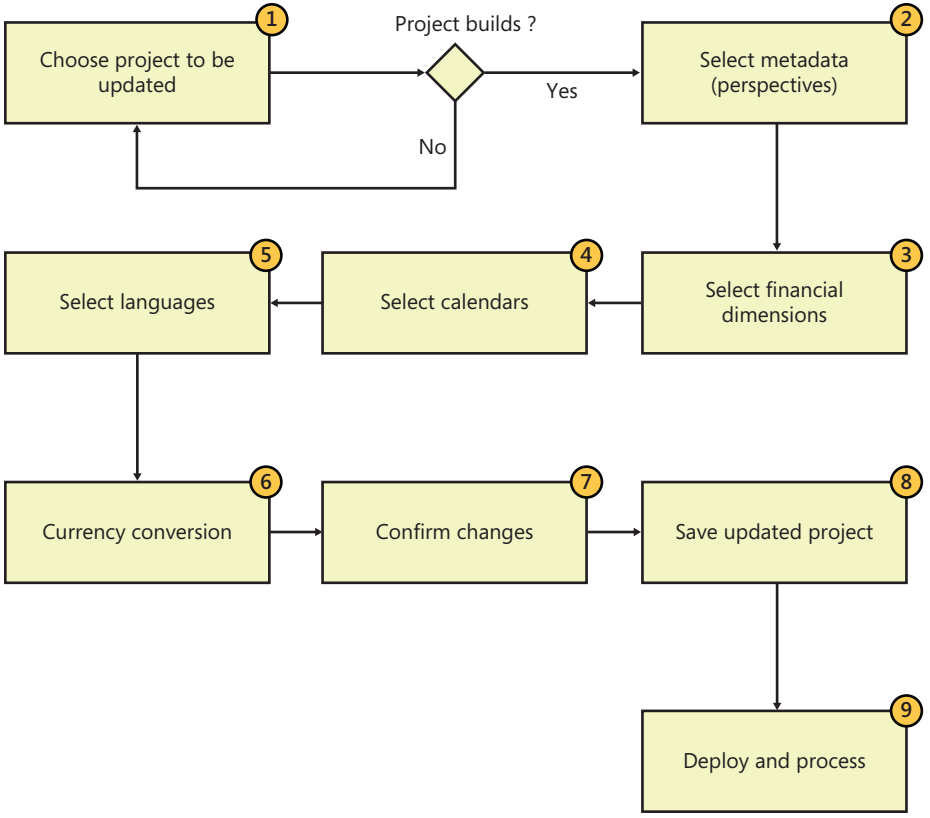


FIGURE 10-10 Updating a cube with the SQL Server Analysis Services Project Wizard.

For example, if you remove the Accounts Receivable perspective from the selection, the Accounts Receivable cube will be removed from the project that you are updating. If you model a new perspective in the AOT and include it in the project, the corresponding measures and dimensions will be created and added to the SSAS project.

For a description of metadata definitions and the resulting analytic artifacts, see "Defining Cubes in Microsoft Dynamics AX" at *http://msdn.microsoft.com/en-us/library/cc615265.aspx*. Metadata is also covered in further detail later in this chapter, in the "Creating cubes" section.

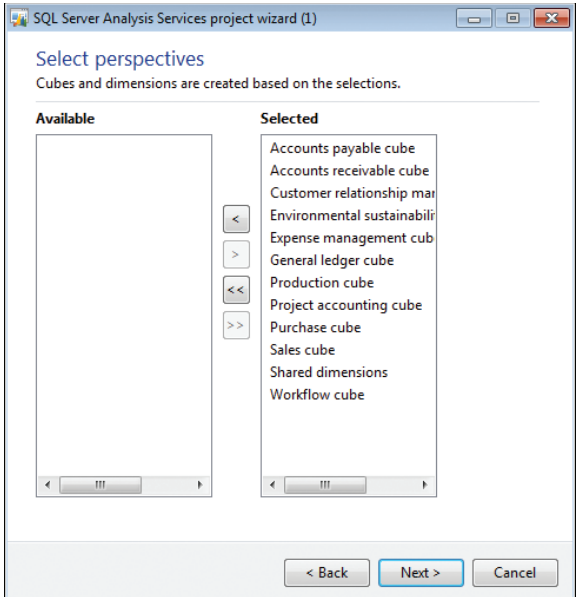


FIGURE 10-11 Selecting metadata.

Select financial dimensions

On the next wizard page, you are prompted to select the Microsoft Dynamics AX financial dimensions to include in the project, as shown in Figure 10-12.

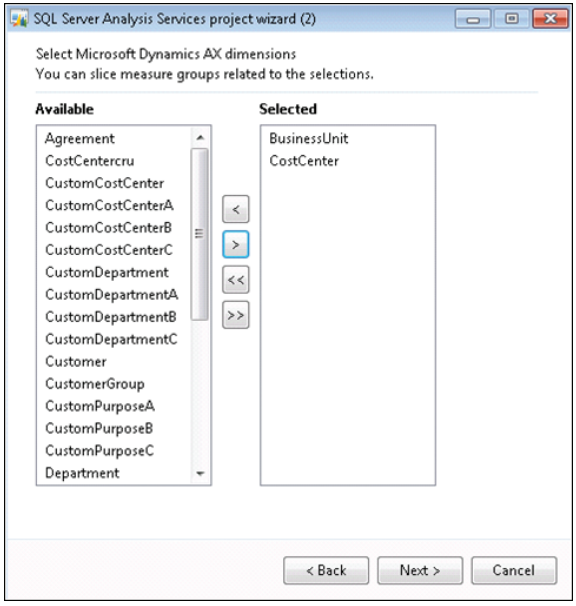


FIGURE 10-12 Selecting financial dimensions.

Each financial dimension that you select is added as an OLAP dimension with the same name. If a dimension by that name already exists within the SSAS project, the system will disambiguate the newly added dimension by adding a suffix.

Select calendars

Next, the wizard prompts you to select the calendars to include as date dimensions, as shown in Figure 10-13. If you have defined any additional calendars, you can include them in the project at this point.

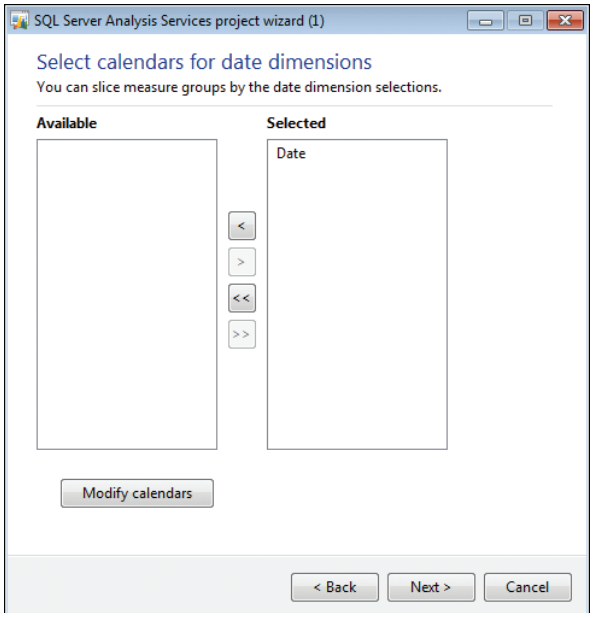


FIGURE 10-13 Selecting a calendar for a date dimension.

In Microsoft Dynamics AX 2009, the prebuilt analysis project included two date dimensions: a Gregorian calendar–based dimension called DATE and a fiscal calendar–based dimension called FISCALPERIODDATEDIMENSION. If you wanted to include additional date dimensions, you would have had to customize the prebuilt project by using Business Intelligence Development Studio.

Microsoft Dynamics AX 2012 includes a utility called Date Dimensions (see Figure 10-14) that lets you define custom calendars for analysis purposes. A default calendar, Date, is included with the product, and you can define additional calendars by using Date Dimensions.

For each calendar that you add on this wizard page, the system creates a date dimension in the SSAS project. For example, if you added a new calendar called Sales Calendar, the system will add a date dimension called Sales Calendar. In addition, the system will create role-playing date dimensions that correspond to each of the dates that are present in cubes. You can't remove the prebuilt date dimension from the project.

You can start Date Dimensions directly from the SQL Server Analysis Services Project Wizard, or from the System Administration area page.

You can define a calendar by selecting the beginning of the year and the first day of the week. For example, for the Sales calendar, the year starts on April 1 and ends on March 31, and the week starts on Sunday. You can enter a date range to specify the calendar records that you want the system to populate in advance. You can also select the hierarchies that will be created for each calendar.

When you close the form, if you added or modified calendars, the system will populate dates according to the new parameters that you defined. In addition, the system will add the required translations. As you will notice later, the system adds a rich set of attributes for each calendar defined here. You can use any of these attributes to slice the data contained in cubes.

In addition, Date Dimensions adds a NULL date record (1/1/1900) and a DATEMAX date record (31/12/2154) to each calendar, so that fact records that contain a NULL date or the DATEMAX date will be linked to these extra records, preventing an "unknown member" error from occurring during cube processing.

Select languages

The prebuilt SSAS project uses EN-US as the default language. However, you might have sites in other countries/regions and want the users there to be able to view measure and dimension names in their own languages.

The project can include additional languages through a feature in SSAS called Translations. The Translations feature enables dimensions, measures, many other kinds of metadata, and data to be translated to other languages by letting you add companion text in other languages.

For example, if you add German translations to the project, when a German user views data in a cube by using, for example, Microsoft Excel, data labels are displayed in German.

The prebuilt SSAS project does not include translated strings. However, translated labels are already available in the system. The SQL Server Analysis Services Project Wizard lets you add any of the required languages to the project by using existing translations from within Microsoft Dynamics AX, as shown in Figure 10-14.

It is recommended that you add only the translations that you need. Each translation adds strings to your project, and the size of the project increases by a few megabytes each time you add a language. In addition, processing gets a bit slower and the size of the backup increases.

If you have the Standard edition of SQL Server 2005 or SQL Server 2008, you could not add additional translations (for Microsoft Dynamics AX 2009). You had to buy the Enterprise edition of SQL Server in order to add translations to cubes. This restriction has been removed in SQL Server 2008 R2 and later versions.

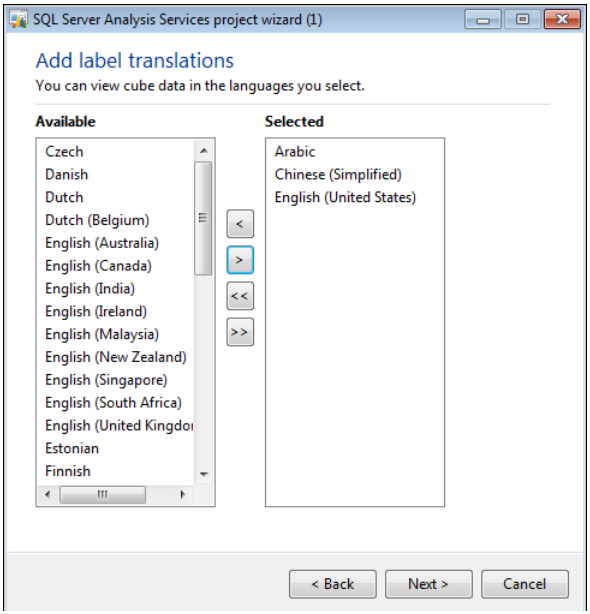


FIGURE 10-14 Selecting languages.

Labels associated with Microsoft Dynamics AX tables and views are carried through to the corresponding dimensions and measures. It is also possible to add specific labels to dimensions and measures by defining the labels in perspectives. For more information, see the section "Define perspectives" later in this chapter.

If you manually add translations to the project in Business Intelligence Development Studio, the wizard overwrites the labels every time you run the Update function, by sourcing labels from Microsoft Dynamics AX. To add your own translations, either define a new label and associate it with the object or change the translation in Microsoft Dynamics AX by using Microsoft Dynamics AX Label Editor.

Add support for currency conversion

The prebuilt SSAS project contains the logic to convert measures that are based on the Microsoft Dynamics AX extended data type (EDT), *AmountMST*, to other Microsoft Dynamics AX currencies. For example, if the amount was recorded in USD, you can display the value of the amount in GBP or EUR by using the analysis currency dimension to slice the amount.

If you want to, you can exclude currency conversions by clearing the check box on the wizard page shown in Figure 10-15.

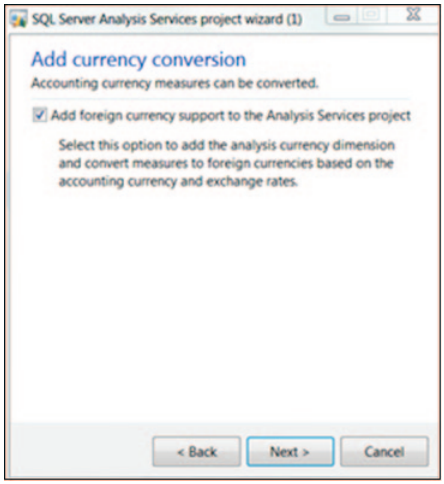


FIGURE 10-15 Selecting support for currency conversion.

Note Removing support for currency conversion not only removes this feature but might also cause prebuilt reports to fail, because they rely on the currency conversion option to be displayed in Role Centers.

For more information about currency conversion, see the section "Add currency conversion logic" in the "Creating cubes" section.

Confirm your changes

When you click Next on the Add Currency Conversion page, the wizard goes to work, performing the following tasks:

- Generates a new project based on the perspectives and other options that you have chosen.
- Compares the newly generated project with the project you wanted to update.
- Displays the differences between the new project (that is, the changes you want to apply) and the old project, as shown in Figure 10-16.

In the wizard, it is assumed that you want to confirm all changes; therefore, all changes are selected by default. If you want the wizard to apply all changes, click Next, and then the wizard will create a project that includes the changes that you selected.

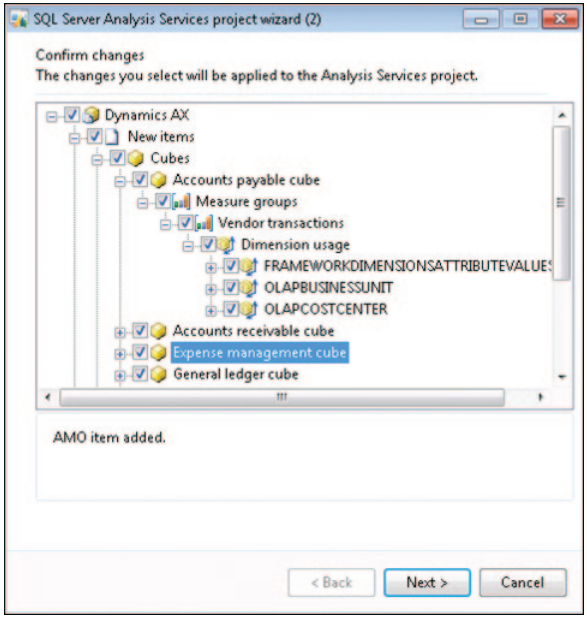


FIGURE 10-16 Confirming changes to an SSAS project.

However, if you are an experienced BI developer and want more granular control of the Update option, you can examine the updates in detail and accept or reject the changes.

Be aware, however, that making changes to the wizard at a granular level may result in inconsistencies within the analysis project. If such inconsistencies result in a project that does not build, the wizard displays a message to inform you.

Here are some examples of when you might want to evaluate changes individually:

- You might have removed some perspectives from the generation process (for example, you have not implemented Project Accounting functionality in Microsoft Dynamics AX and are therefore not interested in the Project Accounting cube). Ordinarily, the system would remove the resulting analytic artifacts, including a dimension. However, you may want to use that dimension in analysis, even if the Project Accounting cube is not used. Therefore, you reject the deletion of that dimension.
- You have added extra attributes to the customer dimension by using Business Intelligence Development Studio. The system would ordinarily delete these extra attributes, because they are not associated with Microsoft Dynamics AX metadata. However, you may want to reject the deletion and keep these extra attributes intact.

Tip If you make too many customizations directly within BI Development Studio, the wizard detects a large number of changes. You must then review each change and approve or reject it. At some point, running the wizard to update the project may cause too much overhead. Therefore, if you are an experienced BI developer, and you have customized the prebuilt project extensively within Business Intelligence Development Studio, don't use the Update function again. Instead, maintain your project in Business Intelligence Development Studio.

Save the updated project

Next, the wizard applies the changes you specified in the previous step. If you simply clicked Next (that is, you did not make any changes to the options selected by the wizard), the wizard would save the resulting project.

If you made changes and the wizard encountered inconsistencies (that is, the project is in an error state and does not build), it displays a warning asking whether you want to save the project or go back to the confirmation step and reconsider the changes.

If you choose to save the project in an inconsistent state (if you are an experienced BI developer, you might choose this approach), you must fix the project by using Business Intelligence Development Studio; otherwise, subsequent deployment steps will be unsuccessful.

Deploy and process cubes

Next, you can deploy the cubes to an SSAS server and, optionally, process the cubes. As discussed in the "Deploy cubes" section earlier in this chapter, in a multiple-partition environment in Microsoft Dynamics AX 2012 R2, the system will deploy the project to multiple SSAS databases.

Extend cubes

As discussed earlier in this chapter, you can customize the prebuilt analysis project relatively easily by using the SQL Server Analysis Services Project Wizard. But in some cases, you may want to make deeper customizations. For example, you might want to:

- Create a rich hierarchy, such as a parent/child hierarchy to model organizational units.
- Add new KPIs.
- Bring external data into the analysis project and create a custom dimension.

You can use Business Intelligence Development Studio to make these types of changes.

Because the prebuilt BI components are included in the AOT as an SSAS project, you can modify the project. To modify the prebuilt Analysis Services project, do the following:

1. In the AOT, expand the *Visual Studio\Analysis Services Projects* node.
2. Right-click the project that you want to modify, and then click Edit.

   An Infolog message appears, stating that a copy of the SSAS project has been created and saved, as shown in Figure 10-17.

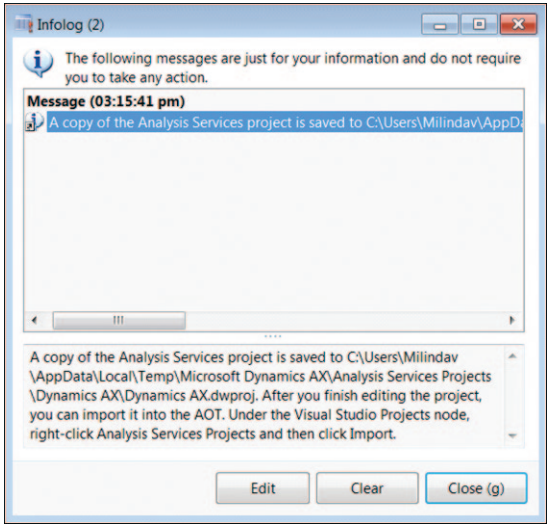


FIGURE 10-17 Infolog message displaying the location of the SSAS project.

If SQL Server Business Intelligence Studio is installed, it will start and open the copy of the project. Changes that you make to the project are not automatically saved to the AOT. You need to save the project and import it back into the AOT.

Figure 10-18 shows the prebuilt SSAS project in Business Intelligence Development Studio.

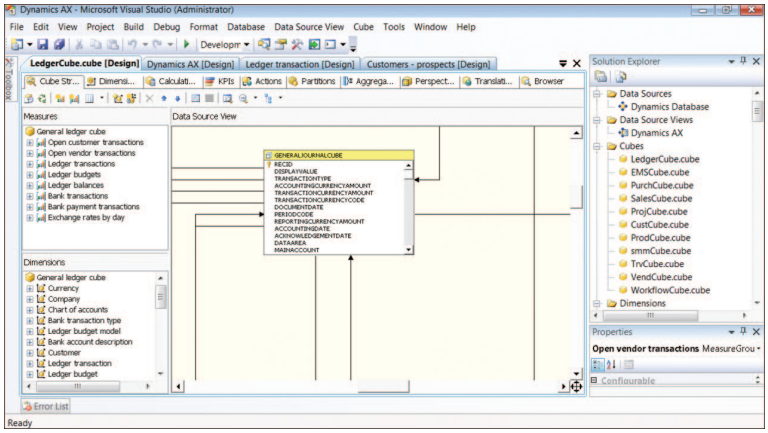


FIGURE 10-18 Dynamics AX SSAS project.

The following sections describe the components of the project.

DSV

The DSV contains the table and view definitions that are used by analytic artifacts. Notice that the OLAP framework has implemented several query definition patterns in the DSV:

- Financial dimensions that the wizard has added appear as custom query definitions in the DSV.
- The OLAP framework has created query definitions corresponding to Microsoft Dynamics AX views.
- The OLAP framework has added a reference relationship to resolve virtual companies, if your Microsoft Dynamics AX installation has virtual company definitions.
- The OLAP framework has created views that make Microsoft Dynamics AX enumerations accessible in all of the languages that have been added to the project.

Avoid modifying any of the framework-generated objects in the DSV. Any changes that you make to these objects are overwritten without warning the next time you update the project.

You may add your own objects to the DSV (for example, new query definitions. The Project Update option will preserve these objects.

In Microsoft Dynamics AX 2012 R2, do not implement any partition-specific logic in any of the query definitions. Otherwise, when the project is deployed to multiple partitions, the system may generate processing errors. (Because the framework adds partition-specific logic to the DSV at deployment time, it may not apply the changes accurately to your query definitions.)

Data source

A data source has been created that points to the Microsoft Dynamics AX OLTP database.

Dimensions, measures, and measure groups

In Figure 10-18, notice the dimensions that are included with the Microsoft Dynamics AX 2012 prebuilt BI solution, as well as the measures and measure groups.

For a list of measures and dimensions, see "Cube and KPI reference for Microsoft Dynamics AX 2012" at *http://msdn.microsoft.com/en-us/library/hh781074.aspx*.

KPIs and calculations

The SSAS project contains prebuilt KPIs and calculations. Microsoft Dynamics AX 2012 does not provide the capability to model KPIs and calculations in the AOT. You can modify these definitions or add new ones directly in Business Intelligence Development Studio.

Integrate Microsoft Dynamics AX analytic components with external data sources

Data warehouses are a popular solution for providing analytic capabilities to users. Until recently, data warehouses were the only reasonable solution for building robust analytic capability. However, as applications become easily interoperable and as technologies such as in-memory databases and OLAP become cost-effective and simpler to use, building a data warehouse is not the only solution to meet analytic requirements.

Table 10-2 presents several architecture options for integrating external data with the prebuilt analytic solution; a data warehouse is just one of the options. The columns represent architecture options, whereas the rows represent the benefits and cost implications of each option.

TABLE 10-2 Options for integrating external data with Microsoft Dynamics AX for analysis.

	Data mash-ups	Integration into Microsoft Dynamics AX	SSAS–based integration	ETL–based data warehouse
Architecture	Tools such as Excel PowerPivot let users mash up and report on data.	Bring external data into Microsoft Dynamics AX tables by using services or data import jobs.	Integrate external data into cubes by using capabilities in SSAS.	Extract, transform, and load (ETL) Microsoft Dynamics AX and other data into a data warehouse instance.
Key benefit or capability	Ad-hoc and user driven	Uses Microsoft Dynamics AX tools and interfaces to analyze and report on data.	Uses capabilities within SSAS to incorporate external data into prebuilt cubes.	Offers complex integration capabilities. Patterns and processes are widely understood.
Complexity	Low – Use of client tools	Medium – Use of Microsoft Dynamics AX tools	Medium – Localized modifications to prebuilt cubes	High
Cost	Low – User-driven	Moderate	Moderate	High
Time to implement	Low	Medium	High	Very high
Expertise needed	Low	Moderate	Moderate	High

When most data is in Microsoft Dynamics AX (assuming that Microsoft Dynamics AX is the predominant source of data in the organization), you have two options.

The data mash-up option is best suited to an environment where capable users author and publish analyses for the use of others. This option relies on client tools such as Excel PowerPivot. Microsoft Dynamics AX 2012 enables Microsoft Dynamics AX queries to be published to data mash-up tools through OData feeds, or as data exports to Excel.

You can bring external data into Microsoft Dynamics AX either through services (data services consumed by means of inbound ports) or as batch jobs that are executed periodically to import data into tables. With this approach, external data is represented as read-only data within Microsoft Dynamics AX. The benefit to this approach is that external data appears as native Microsoft Dynamics AX data to Microsoft Dynamics AX tools. You can create analytics, reports, and inquiry forms that use the combined data.

A more complex approach involves integrating external data directly into the prebuilt BI solution. With this option, a BI developer adds another data source to the prebuilt BI solution by using Business Intelligence Development Studio. Additional data tables are brought into the DSV by using the new data connection. It is possible to create dimensions and measures by using the new tables in the DSV.

The traditional ETL-based data warehouse option is suited to scenarios that require complex transformations or large volumes of data. Although this option is more flexible in terms of capabilities, it is also the most expensive to implement and manage.

You might want to build a data warehouse to implement the following scenarios:

- **Integrate external data sources with Microsoft Dynamics AX data** In this approach, the Microsoft Dynamics AX implementation serves as one of many corporate applications. Although Microsoft Dynamics AX contains some of the corporate data, other systems contain a considerable portion of the data. To make decisions, data must be combined across systems, and the data warehouse serves that need.
- **Incorporate legacy data into Microsoft Dynamics AX analytics** Most organizations migrate recent data when implementing Microsoft Dynamics AX. Legacy data is still maintained in read-only instances of legacy applications. Although legacy data is no longer used for operational purposes, it is required for historical trend analysis. A data warehouse serves as the repository where legacy data is combined with current data.

Although Microsoft Dynamics AX 2012 does not directly support the creation of a data warehouse schema, the following artifacts generated in Microsoft Dynamics AX 2012 can be used to build a data warehouse:

- The DSV generated as part of the prebuilt analytic solution can be used within SQL Server Integration Services when an ETL package is developed to extract data from Microsoft Dynamics AX.
- Microsoft Dynamics AX document services can be consumed as data sources based on Simple Object Access Protocol (SOAP).
- Microsoft Dynamics AX queries can be exposed as OData feeds.

Creating cubes

This section discusses how to create new cubes and reports by using tools built into Microsoft Dynamics AX 2012.

Figure 10-19 shows the four-step process for creating a new cube.

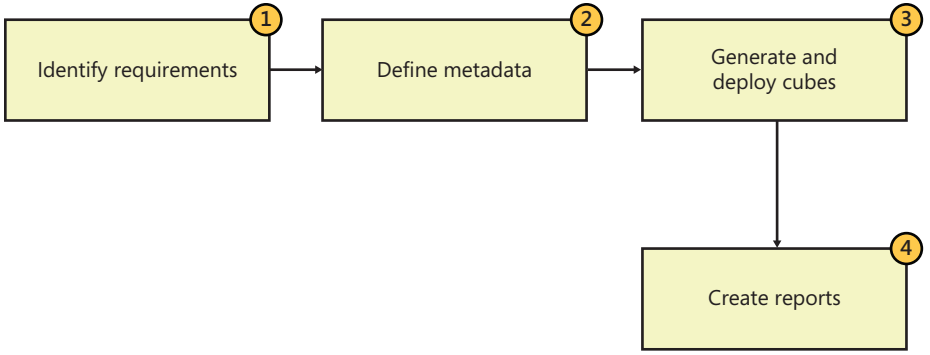


FIGURE 10-19 Creating a new cube.

The following sections describe each step in more detail.

Identify requirements

Often, when a user asks for additional information, you get a request for a new report (or two or three). For example, you might get a requirement request for a report like the one shown in Figure 10-20 from someone in the Sales department.

Sales by channel Report

Sales Channel	January	Februrary	March	April	May	June
Intercompany Customers	73,288.76	1,148.34	47,132.57	7,744.09	67,652.95	
Internet Customers	79,760.78		40,192.50		50,407.77	
Major Customers	181,469.80	283,228.60	546,870.58	389,739.45	258,417.75	446,449.74
Retail Customers	83,064.07	112,599.72	96,921.17	118,955.26	130,211.83	211,698.25
Wholesale Customers	867,567.34	7,173.20	359,152.63	(3,871.87)	619,001.47	7,966.07
Grand Total	**1,285,150.75**	**404,149.86**	**1,090,269.45**	**512,566.93**	**1,125,691.77**	**666,114.06**

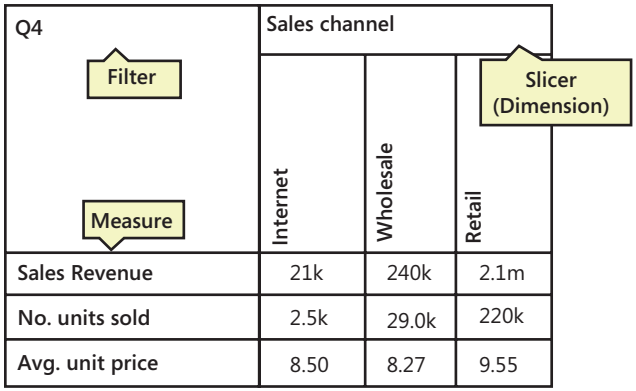
FIGURE 10-20 Sample Sales by channel report.

This report shows sales revenue trends by sales channel. More formally stated, this report shows sales revenue by sales channel by calendar month.

The request for this report might be followed by requests for "a few additional reports." Some of the typical follow up questions would be:

- What about quarterly trends? Is there seasonality?
- Are some regions doing better than others?
- Can we see the number of units sold instead of revenue?
- Can we see the average unit price? Are steep discounts being given?

If you were to build a PivotTable to answer these questions (which is probably a good idea, because this would let the users slice the data, thus saving you from the effort of building all of those reports), you could construct a PivotTable like the one shown in Figure 10-21.



Sales Revenue	21k	240k	2.1m
No. units sold	2.5k	29.0k	220k
Avg. unit price	8.50	8.27	9.55

FIGURE 10-21 Sales PivotTable.

In this case, you have identified the measures (the numbers you are interested in) and the dimensions (the pivots for the data).

The following sections show how to build a cube to meet these requirements.

Define metadata

The next step is to determine which Microsoft Dynamics AX tables or views contain this information. For the purpose of this example, assume the following:

- The CUSTTRANSTOTALSALES view contains sales invoice details.
- The CUSTTABLECUBE view contains master data about customers.
- The CUSTPAYMMODETABLE table contains payment mode information.

Define perspectives

Next, you need to define the metadata that is required to generate the cube in the AOT. As you might recall from Microsoft Dynamics AX 2009, you define the metadata required to generate cubes in the *Data Dictionary\Perspectives* node of the AOT.

Each perspective corresponds to a cube. Tables or views that are contained in a perspective node generate measures or dimensions. Depending on table relationships (and inferred view relationships), measures are associated with dimensions within the generated project.

Note In Microsoft Dynamics AX 2012, you can use views to model a cube.

If you want to designate a perspective node that contains only dimensions, Microsoft Dynamics AX 2012, includes a property at the perspective level specifically for this purpose: *SharedDimension-Container*. If you designate a perspective as a shared dimension container, tables and views within that perspective will be used only to create dimensions. Moreover, all of the dimensions will be associated with all of the measures; that is, they are truly shared dimensions, provided that they are related in Microsoft Dynamics AX.

Follow these steps to create the new perspective for this example:

1. In the AOT, expand the *Data Dictionary\Perspectives* node.
2. Create a new perspective node, and name it ***MyCustomers***.

   The new node contains two subnodes: *Tables* and *Views*.
3. Set the *Usage* property of the node to *OLAP* to designate that this perspective will be used to generate a cube.

   If you are familiar with Microsoft Dynamics AX 2009, you may notice that the *Ad-Hoc Reporting* option for the *Usage* property is missing in Microsoft Dynamics AX 2012. You can select only *OLAP* or *None*. It is no longer possible to generate report models by using perspectives in Microsoft Dynamics AX 2012.
4. Drag the tables and views listed in the previous section into the newly created perspective.

For more information, see "How to: Create a Perspective for a Cube" at *http://msdn.microsoft.com/en-us/library/cc617589.aspx*.

Define table-level properties

Strictly speaking, table-level properties (see Figure 10-22) are optional. However, if you do use them, cubes will perform better.

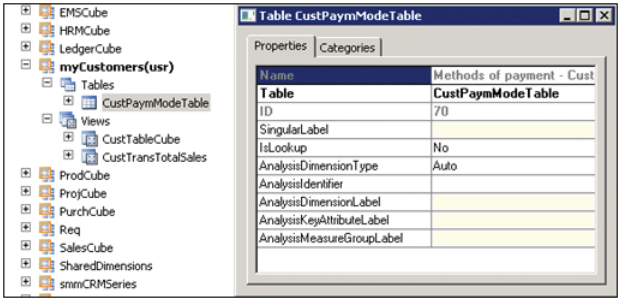


FIGURE 10-22 Table-level properties.

You can also specify custom labels to give specific names to generated measure groups and dimensions. *AnalysisDimensionLabel*, *AnalysisKeyAttributeLabel*, and *AnalysisMeasureGroupLabel* are new properties introduced in Microsoft Dynamics AX 2012. Instead of providing English text, you can provide Microsoft Dynamics AX labels so that dimension names are translated into other languages.

The *AnalysisIdentifier* property defines the field that provides the name for a dimension key. If you look at the *Name* field for this property in Figure 10-22, you will notice that the Methods Of Payment dimension is keyed by the *Name* field.

For more information, see "Business Intelligence Properties" at *http://msdn.microsoft.com/en-us/library/cc519277.aspx*.

If you are a fan of the semantics introduced with the *IsLookUp* property in Microsoft Dynamics AX 2009, you will be pleased to know that views in Microsoft Dynamics AX 2012 provide this functionality. However, the *IsLookUp* property will be deprecated in future releases, so it is recommended that you do not use this property.

Define field-level properties

Defining field-level properties is the key step in defining metadata. You need to identify individual measures and attributes that are necessary in the cube.

First, expand the CUSTTRANSTOTALSALES view, and set the field properties as shown in Table 10-3.

TABLE 10-3 Field-level property settings for the sales report example.

Field	*AnalysisUsage*	*AnalysisDefaultTotal*	*ExchangeRateDateField*
AmountMST	*Measure*	*Sum*	*TransDate*
TransType	*Attribute*	*Auto*	
TransDate	*Attribute*	*Auto*	
All others	*Auto*	*Auto*	

The *AmountMST* field will generate a measure that is summed when it is aggregated. *ExchangeRateDateField* is a new attribute added in Microsoft Dynamics AX 2012 for currency conversion. In this example, the OLAP framework should convert the *AmountMST* measure to all available currencies, so that users can analyze transactions (possibly conducted in different currencies) across a common currency. The *TransDate* field contains the date on which the measure will be converted into other currencies with Microsoft Dynamics AX exchange rates.

Users need to be able to slice the data by *TransType* and *TransDate*, so these fields are designated as attributes.

Next, open the CUSTTABLECUBE view, and set the field-level properties as shown in Table 10-4.

TABLE 10-4 Field-level properties for the CUSTTABLECUBE view.

Field	AnalysisUsage	AnalysisDefaultTotal
AccountNum	*Measure*	*Count*
Blocked	*Attribute*	*Auto*
GroupName	*Attribute*	*Auto*
City	*Attribute*	*Auto*
County	*Attribute*	*Auto*

Field	AnalysisUsage	AnalysisDefaultTotal
Name	*Attribute*	*Auto*
State	*Attribute*	*Auto*
MainContactWorker	*Attribute*	*Auto*
All others	*Auto*	*Auto*

Finally, expand the CUSTPAYMODE table, and set the field-level properties as shown in Table 10-5.

TABLE 10-5 Field-level properties for the CUSTPAYMODE table.

Field	AnalysisUsage	AnalysisDefaultTotal
Name	*Attribute*	*Auto*
PaymMode	*Attribute*	*Auto*
TypeofDraft	*Attribute*	*Auto*
AccountType	*Attribute*	*Auto*
All others	*Auto*	

For more information about field-level properties, see "Business Intelligence Properties" at *http://msdn.microsoft.com/en-us/library/cc519277.aspx*.

Generate and deploy the cube

After you define the necessary metadata, you can generate an SSAS project by using the SQL Server Analysis Services Project Wizard. You can deploy and process the project directly from the wizard, or you can open the project in BI Development Studio and extend it by using SQL Server functionality.

Define the project

In the wizard, select the Create option, because you are creating a new project, and provide a name. Alternatively, if you want to include the new cube in the prebuilt SSAS project, you can select the Update option.

On the next page, select the perspectives that are used to generate cubes and dimensions within the project. For this example, you would select the MyCustomers perspective. You can include one or more perspectives within the same project.

You can also include Microsoft Dynamics AX financial dimensions, in addition to Microsoft Dynamics AX calendars and Microsoft Dynamics AX languages, as discussed earlier in this chapter.

Add currency conversion logic

Next, the wizard lets you add currency conversion logic to the project.

As you may recall, while defining field-level properties for the perspective, *AmountMST* was identified as a measure that needs to be converted to other currencies. The *AmountMST* field contains an amount that is recorded in the accounting currency of the company. Because

Microsoft Dynamics AX might contain multiple companies that have different accounting currencies, transactions might be recorded in different accounting currencies.

For example, the CEU company's accounting currency is GBP, whereas the CEUE company's accounting currency is USD. In the *AmountMST* field, sales for CEU are recorded in GBP, whereas those for CEUE are recorded in USD.

Because a cube aggregates data across companies, a user browsing the cube could inadvertently add GBP values to USD values unless something is done to differentiate the two amounts. The Microsoft Dynamics AX 2012 OLAP framework builds this mechanism for you in the form of currency conversion support.

Microsoft Dynamics AX 2012 cubes contain two system dimensions: Currency and Analysis Currency. If the user uses the Currency dimension to split the measures that are shown, Microsoft Dynamics AX displays amounts only in the chosen currency. If the user uses the Analysis Currency dimension to split the measures that are shown, all amounts are shown, but the resulting values are converted to the chosen analysis currency by using Microsoft Dynamics AX exchange rates. This happens through currency conversion.

Here is an example: assume that the transactions shown in Figure 10-23 are included in the CUSTTRANSTOTALSALES view. (Note that two columns have been added, *Accounting Currency* and *AmountCur*, to clarify that each company has a different accounting currency.)

Rec ID	Company	Accounting Currency	Trans Date	Amount Cur	Amount MST
1	CEE	USD	1/1/2012	21,000 (JPY)	202 (USD)
2	DMO	CAD	1/1/2012	300 (GBP)	475 (CAD)
3	CEU	GBP	2/1/2012	300 (GBP)	300 (GBP)

FIGURE 10-23 Transactions for companies in different accounting currencies.

If a user creates a PivotTable and displays the total *AmountMST* value split by the Analysis Currency dimension, the result is as shown in Figure 10-24.

	Analysis Currency		
	USD	CAD	GBP
Amount MST	1058	1079	813

FIGURE 10-24 Analysis currency.

To get the value of *AmountMST* in USD, the system calculated the USD equivalent of each of the amounts, as shown in Figure 10-25.

= Amount MST in Analysis Currency

= ∑ Amount MST in Accounting Currency x Exchange Rate

= 202 x 1.0000 + 475 x 0.9800 + 300 x 1.3000

= 202 + 466 + 390

= 1, 058

CAD to USD exchange rate on 1/1/2012

GBP to USD exchange rate on 2/1/2012

FIGURE 10-25 Currency conversion for analysis.

To determine the exchange rate between CAD and USD, and between GBP and USD, the system used the field-level metadata tag *ExchangeRateDateField*. For this example, the *ExchangeRateDateField* value for *AmountMST* is *TransDate*. So the *TransDate* value associated with each record was used to find the exchange rate to use for the conversion.

Microsoft Dynamics AX 2012 has the concept of a rate type. In other words, multiple exchange rates can be associated with a given company. A company can use different rates for different purposes or different rates for different locations. The Microsoft Dynamics AX 2012 OLAP framework uses the system exchange rate type for the currency conversion logic. This rate type is a systemwide parameter that a system administrator specifies on the System Parameters form (System Administration > Setup > System Parameters), as shown in Figure 10-26.

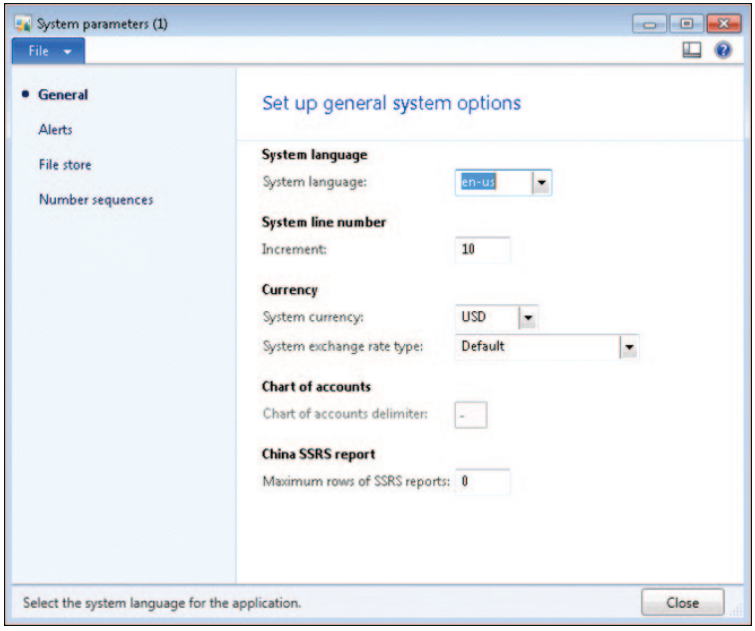


FIGURE 10-26 Setting the system currency and exchange rate type.

If you create a PivotTable with the Currency dimension, *AmountMST* values are filtered by the specified currency, as shown in Figure 10-27. You would expect this behavior if you created a PivotTable with any dimension.

	Currency		
	USD	CAD	GBP
AmountMST	202	475	300

FIGURE 10-27 PivotTable with the Currency dimension.

Provided that you define the field-level metadata tag *ExchangeRateDateField*, the wizard adds the currency conversion calculation to the generated project as a multidimensional expression (MDX) script. The wizard also adds the system dimension Analysis Currency (the Currency dimension is added regardless of whether you select currency conversion). The wizard also creates an intermediate measure group called *Exchange Rates By Day* in each cube.

If you open the generated project in Business Intelligence Development Studio, you can see the currency conversion calculation created by the wizard:

```
CALCULATE;
//------------------------------------------------------------------------------
// Dynamics AX framework generated currency conversion script.
// Customizing this portion of the script may cause problems with the updating
// of this project and future upgrades to the software.
//------------------------------------------------------------------------------
Scope ( { Measures.[Amount] } );
    Scope( Leaves([Exchange rate date]),
           Except([Analysis currency].[Currency].[Currency].Members,
                  [Analysis currency].[Currency].[Local]),
           Leaves([Company]));
        Scope( { Measures.[Amount] } );
            This = [Analysis currency].[Currency].[Local] * ((Measures.[Exchange rate],
StrToMember("[Currency].[Currency].&["+[Company].[Accounting currency].CurrentMember.Name+"]"))
/ 100.0);
        End Scope;
    End Scope;
    Scope( Leaves([Exchange rate date]),
           Except([Analysis currency].[Currency name].[Currency name].Members,
                  [Analysis currency].[Currency name].[Local]),
           Leaves([Company]));
        Scope( { Measures.[Amount] } );
            This = [Analysis currency].[Currency].[Local] * ((Measures.[Exchange rate],
StrToMember("[Currency].[Currency].&["+[Company].[Accounting currency].CurrentMember.Name+"]"))
/ 100.0);
        End Scope;
    End Scope;
    Scope( Leaves([Exchange rate date]),
           Except([Analysis currency].[ISO currency code].[ISO currency code].Members,
                  [Analysis currency].[ISO currency code].[Local]),
```

```
            Leaves([Company]));
        Scope( { Measures.[Amount] } );
            This = [Analysis currency].[Currency].[Local] * ((Measures.[Exchange rate],
StrToMember("[Currency].[Currency].&["+[Company].[Accounting currency].CurrentMember.Name+"]"))
/ 100.0);
        End Scope;
    End Scope;
    Scope( Leaves([Exchange rate date]),
           Except([Analysis currency].[Symbol].[Symbol].Members,
                  [Analysis currency].[Symbol].[Local]),
           Leaves([Company]));
        Scope( { Measures.[Amount] } );
            This = [Analysis currency].[Currency].[Local] * ((Measures.[Exchange rate],
StrToMember("[Currency].[Currency].&["+[Company].[Accounting currency].CurrentMember.Name+"]"))
/ 100.0);
        End Scope;
    End Scope;
End Scope;
//--------------------------------------------------------------------------------
// End of Microsoft Dynamics AX framework generated currency conversion script.
//--------------------------------------------------------------------------------
```

This logic is similar to the code added by the Define Currency Conversion option in the SSAS Business Intelligence Wizard. If the selected Microsoft Dynamics AX exchange rate type does not have records corresponding to the dates (for example, *TransDate*) that are present in data, the calculations will use the most recent rate for the corresponding currency pair.

Important The wizard maintains this script as you configure and update analysis projects. If you modify the script manually, your changes will be overwritten by the framework each time.

Save the project

After you specify currency conversion options, the system will generate the project and prompt you for a destination to which to save the project.

You can save the project in the AOT or on disk. This gives you the flexibility to maintain SSAS projects in the development environment of your choice. OLAP framework tools, such as the SQL Server Analysis Services Project Wizard, will work with projects whether they are on disk or in the AOT.

If you save the project in the AOT, the project will be saved in your layer.

Deploy and process the project

You can deploy the project directly to the Analysis Services server at this stage. It's important to note that the wizard calls the Analysis Services deployment functionality behind the scenes. If you do not have the Microsoft Dynamics AX Development Workspace (including Business Intelligence Development Studio) installed on your computer, this step may fail.

As discussed earlier, in Microsoft Dynamics AX 2012 R2 you can deploy a project to multiple partitions. If you have multiple partitions defined, you can deploy the project to the set of partitions you choose.

Add KPIs and calculations

You can define KPIs by using Business Intelligence Development Studio after you generate the project. You implement KPIs and calculated measures by using MDX.

The KPIs and calculated measures in the prebuilt SSAS project are also created in this way. If you create your own KPIs and calculated measures, the SQL Server Analysis Services Project Wizard will preserve them when you perform updates.

For more information, see "Walkthrough: Defining KPIs for a Cube" at *http://msdn.microsoft.com/en-us/library/dd261469.aspx*.

If you are an expert MDX developer, you might be tempted to implement complex calculations and KPIs. However, a best practice is to move your calculations to Microsoft Dynamics AX views and tables as much as possible. This way, you not only use the expressive power of Microsoft Dynamics AX, but you also move the calculations that must be pre-aggregated, so that you get better run-time performance.

You can move calculations to Microsoft Dynamics AX in the following ways:

- **Reuse Microsoft Dynamics AX tables and fields** Chances are that the Microsoft Dynamics AX schema already contains most of the calculations that you need. If the information is not directly available in the primary table, review secondary tables and fields to see if corresponding fields are available. A small investment in reviewing the schema will save you a lot of MDX code.
- **Define Microsoft Dynamics AX views with computed columns** Microsoft Dynamics AX 2012 view support in perspectives enables a host of scenarios where multiple tables can be joined to create rich views. The Microsoft Dynamics AX 2012 view framework also provides support for creating computed columns in Microsoft Dynamics AX views. For more information, see "Walkthrough: Add a Computed Column to a View" at *http://msdn.microsoft.com/en-us/library/gg845841.aspx*.

Displaying analytic content in Role Centers

After you create a cube, users can navigate through the aggregated measures and slice them on the dimensions. This section describes ways that you can expose cube content to users.

However, before discussing the presentation tools, this section examines the jobs that people actually do in an organization to help, you understand the nature of the insights that those people need to do those jobs better.

Table 10-6 lists some options for exposing cube data. Later sections discuss those options in greater detail.

TABLE 10-6 Ways of exposing cube data to users.

Option	Capability	Author	Additional requirements
SQL Server Power View	Explore data visually and interactively. Create high-quality presentations with data.	Casual user	Microsoft SharePoint Enterprise edition with SQL Server 2012
Excel PivotTables	Analyze data. Slice and dice data by using dimensions.	Power user	SharePoint Enterprise edition with Excel Services
KPIs with the Business Overview web part	Build simple scorecards on Role Centers by adding and removing KPIs and measures.	Power user	
Reports built by using SQL Server Report Builder v3 (SQL RB3)	Build graphical reports by using aggregate data.	Power user	SQL Server Report Builder, SSRS web parts
Reports built by using Microsoft Visual Studio tools for Microsoft Dynamics AX	Build parameterized production reports by using aggregate data.	Developer	
Interactive charts built by using Microsoft Dynamics AX charting controls	Build interactive charts by using aggregate data.	Developer	Microsoft Dynamics AX 2012 R2

Provide insights tailored to a persona

For the purposes of this discussion, the people in an organization, or personas, are divided into three broad categories: operational, tactical, and strategic.

- Operational personas, such as an Accounts Receivable administrator, focus primarily on staying productive and performing day-to-day tasks, such as keeping tabs on receivables.
- Tactical personas, such as heads of departments and supervisors, have an additional responsibility as people and resource managers; they need to ensure that their teams function smoothly.
- Strategic personas such as chief executive officers (CEOs) need to take a broader corporate view; they tend to operate on established goals and milestones that are evaluated on a wider scale.

Of course, there is an element of operational focus in a tactical persona, and vice versa, but for simplicity, those aspects are not covered here.

Consider a day in the life of an Accounts Receivable (AR) administrator. Like many AR administrators, this administrator is extremely busy at the end of each month (or every Friday, depending on the natural cycle of the business), calling customers and following up on payments. In this case, the AR administrator focuses on exceptions (large payments that are late). If he has more than a few items to work with, he needs a way to prioritize and filter the cases—or even better—see trends within the items at hand. After he identifies a case, he needs to take action and complete the task; for example, he makes a call or sends a note to ensure that the bill is paid.

In this example, insights would help the AR administrator in three areas:

- First, he needs to detect exceptions.
- Next, he needs to identify clusters, trends, and anomalies.
- Finally, he needs to be able to take action.

Of course, real-world AR administrators don't necessarily follow these steps in succession. But these are three situations where insights need to be applied to help the AR administrator accomplish his daily goals.

Choose a presentation tool based on a persona

Depending on the focus of the persona, different tools and approaches may be necessary.

Table 10-7 shows a list of situations in which each persona requires BI tools to provide insight and suggests presentation tools that would meet the needs of each situation.

TABLE 10-7 Business objectives and tools by persona.

BI requirement	Operational persona	Tactical persona	Strategic persona
Detect exceptions	**Objective:** Track exceptional transactions **Tools:** Cues, Info Parts	**Objective:** Identify abnormal trends, outliers **Tools:** Cues, KPIs	**Objective:** Identify goals that have not been met, identify long-term trends that are not meeting expectations **Tools:** KPIs
Identify clusters and trends	**Objective:** Perform simple analysis (prioritizing, filtering) **Tools:** List pages, AutoReports	**Objective:** Slice aggregated data, prepare sample data and audits **Tools:** Excel, SQL Server Power View, AutoReports	**Objective:** View details, compare, and benchmark with peers and previous results **Tools:** Excel, Business Overview web part, PerformancePoint scorecards
Take action	**Objective:** Seamlessly access detailed data **Tools:** Microsoft Office templates, list pages	**Objective:** Communicate and share patterns, take proactive or corrective action **Tools:** Office templates, SSRS Report Builder 3.0, Management Reporter	**Objective:** Perform reorganizations, start programs, and implement action plans **Tools:** KPIs, Business Overview web part, PerformancePoint scorecards

The tools in Table 10-7 are just suggestions for how you can provide insights to users. However, nothing prevents you from using, for example, the Business Overview web part in a Role Center for an operational persona, or from using cues to display detailed data in a Role Center for a strategic persona. For more information about cues and info parts, see Chapter 5, "Designing the user experience."

SQL Server Power View reports

SQL Server Power View is an interactive, browser-based data exploration, visualization, and presentation tool for casual users that is included in SQL Server 2012.

Power View can be integrated with Microsoft Dynamics AX in several ways:

- Users can use the PowerPivot add-in for Excel to create models and reports that combine Microsoft Dynamics AX data with external data sources. These models are commonly known as *data mash-up applications*. Microsoft Dynamics AX queries exposed as OData feeds are the best means of consuming data with this approach because OData feeds ensure that Microsoft Dynamics AX security is enforced at the AOS level. With PowerPivot, a user simply assembles the data required for the analysis with the help of a PowerPivot designer. After the data is assembled, the user can browse the data by using Excel PivotTable functionality.

  PowerPivot models can be saved to a server running SharePoint Services Enterprise edition. Saved PowerPivot models can be explored with Power View.

- As a developer, you can create tabular models by using SQL Server Data Tools, the Visual Studio–based developer tools for creating BI models. When creating tabular models, you can either start from a PowerPivot model created by a user (that is, create a production version of an existing model) or start from scratch. With either approach, you can create a tabular model that consumes data from Microsoft Dynamics AX by means of OData feeds or cubes.

  After you develop a tabular model, you deploy it to the SSAS server; however, the server must be configured in tabular mode, not multidimensional mode.

Note Starting with SQL Server 2012, an SSAS server can be configured for either multidimensional mode (required for hosting Microsoft Dynamics AX cubes) or tabular mode (required for hosting tabular models). An SSAS server that is in multidimensional mode cannot host a tabular model, and vice versa.

- A system administrator can create a Reporting Services data connection file (.rsds file) for tabular mode. After the data connection has been created, users can explore tabular models by using SQL Server Power View.

Expose a Power View report in a Role Center

Embedding an existing Power View report in a Microsoft Dynamics AX Role Center is easy: just use the Page Viewer web part in SharePoint Server, and enter the URL of the Power View report.

Start the Power View report viewer in a browser window, copy the URL for the report (see Figure 10-28), and then paste it into Notepad.

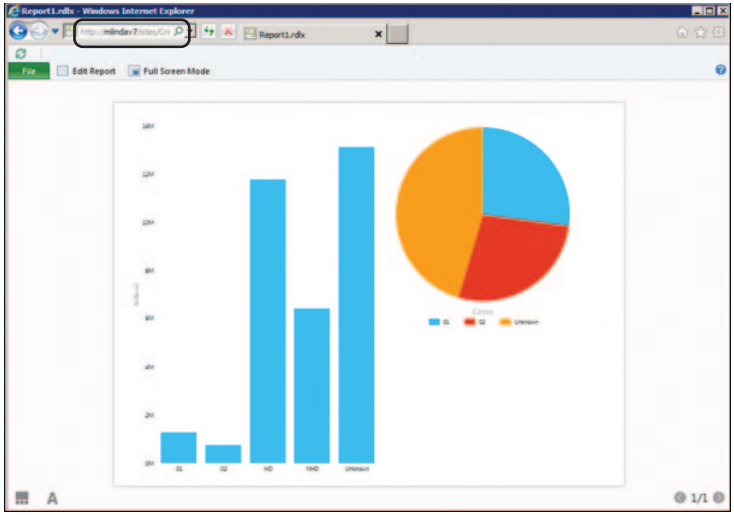


FIGURE 10-28 Power View report.

Here's an example URL:

http://vsqlbuvh0301/_layouts/ReportServer/AdHocReportDesigner.aspx?RelativeReportUrl=/Shared%20Documents/Dynamics-SalesbyRegion.rdlx&ViewMode=Presentation&Source=http%3A%2F%2Fvsqlbuvh0301%2FShared%2520Documents%2FForms%2FAllItems%2Easpx&DefaultItemOpen=1

Notice that the first part of the URL contains the path to the Power View designer and the report being viewed in the designer. The remainder is a collection of parameters that are passed to the designer when it is started by the caller.

You can customize the appearance of the Power View window shown in the Role Center by manipulating these parameters. Table 10-8 lists the parameters and describes what they do.

TABLE 10-8 Power View URL parameters.

Parameter	Description	Suggested value for a Role Center
ViewMode	Defines whether the report is displayed in presentation mode or edit mode.	*Presentation*—Shows the report without edit buttons and the field selection.
Fit	Defines how the contents of the report fit into the window you have chosen.	*True*—Hides the frame around the report.
PreviewBar	Defines whether the preview bar (including Full Screen and Edit buttons) is displayed on the screen.	*False*—Hides the preview bar.
AllowEditViewMode	Defines whether the user can edit the report within the window.	*False*—Makes the report static within the window.
BackgroundColor	Defines the background color if the report doesn't fit into the window. (Not applicable if you want the report to fit into the window.)	*White*—Displays the report in the Role Center without a border.

If you change the URL by applying the parameter values in Table 10-8, the modified URL looks as follows:

http://vsqlbuvh0301/_layouts/ReportServer/AdHocReportDesigner.aspx?RelativeReportUrl=/Shared%20Documents/Dynamics-SalesbyRegion.rdlx&ViewMode=Presentation&Source=http%3A%2F%2Fvsqlbuvh0300%2FPPSubSite%2FShared%2520Documents%2FForms%2FAllItems%2Easpx&DefaultItemOpen=1&Fit=True&PreviewBar=False&BackgroundColor=White&AllowEditViewMode=False

Now, open the Role Center, and select the option to modify or personalize the page. In Edit mode, select Add Web Part, and the Web Part gallery will appear. Select the Page Viewer web part from the gallery of available web parts, as shown in Figure 10-29.

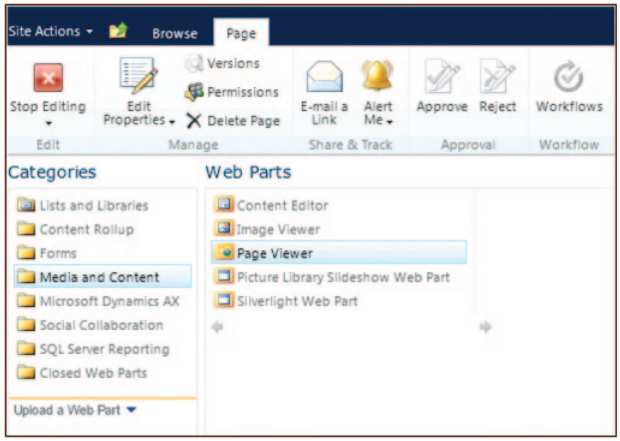


FIGURE 10-29 Web Part gallery.

After you add the web part, specify the URL for the report. You can also provide a friendly title and height and width parameters to suit the window, as shown in Figure 10-30.

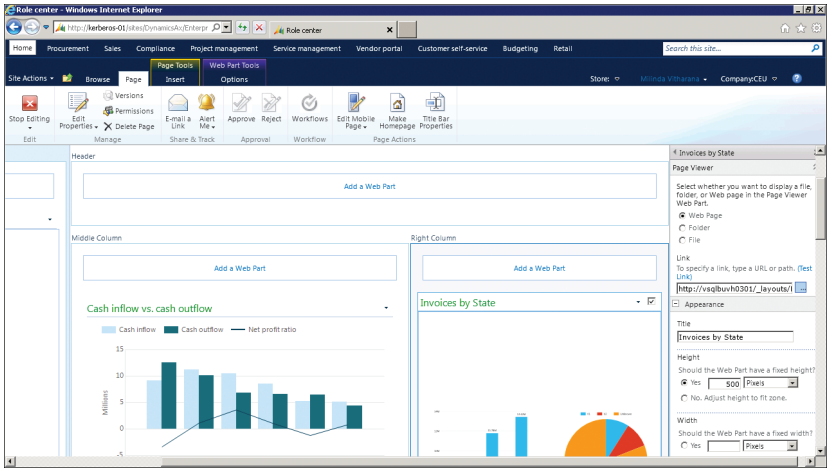


FIGURE 10-30 Specify the presentation options for a Power View report.

Figure 10-31 shows the result. Notice that you can match the color scheme of the Power View report with the color scheme of other reports and charts on the page. This way, users won't notice a difference between the Power View report and the other charts on the page.

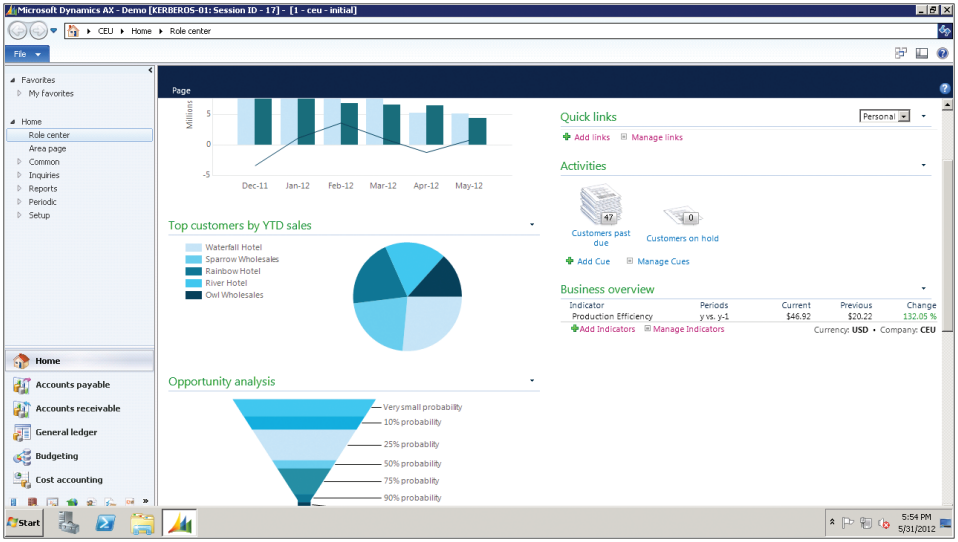


FIGURE 10-31 Power View report in a Role Center.

Allow users to edit a Power View report

In the previous example, the Edit button and the chrome were disabled because an editing experience in a small window would not be optimal. Also, the capability to edit a report may be beyond the reach of some of the users.

However, you can easily allow users to edit a report by creating a quick link to start Power View in a separate browser window.

To do so, create a new URL quick link by clicking the Add Links option in the Quick Links web part, as shown in Figure 10-32.

FIGURE 10-32 Adding a link to the Quick Links web part.

In the Add Quick Link dialog box, paste the URL of the Power View report, as shown in Figure 10-33.

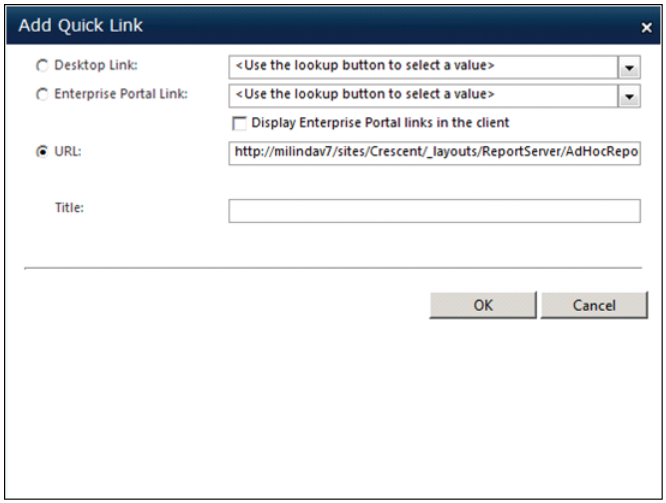


FIGURE 10-33 Specifying a URL to a Power View report.

You will now see the new quick link added, as shown in Figure 10-34.

FIGURE 10-34 A quick link to a Power View report.

Excel

Excel is a simple, yet powerful way to share reports with users in Role Centers. For example, you can:

- Analyze cube data in Excel and create PivotTables.
- Save PivotTable reports to Excel Services for SharePoint.
- Expose Excel worksheets that are saved to Excel Services for SharePoint by using either the Excel Services web part or using the Excel Web App.

For step-by-step instructions that show how to create a PivotTable by using the prebuilt General ledger cube, see "Walkthrough: Analyzing Cube Data in Excel" at *http://msdn.microsoft.com/en-us/library/dd261526.aspx*.

Excel Services for SharePoint 2010, in combination with Excel 2010, let you expose charts and PivotTables built by using the Excel Services REST application programming interface (API). The URL that you obtain by using the REST API can be used to display a chart or a table in Role Centers. For more information about Excel Services, see "Excel Services overview" at *http://technet.microsoft.com/en-us/library/ee424405.aspx*.

Business Overview web part and KPI List web part

The Business Overview web part was introduced in Microsoft Dynamics AX 2009 to display the KPIs in prebuilt cubes in Role Centers. This web part was initially modeled on the KPI List web part in SharePoint Enterprise edition, but it has evolved into a distinct Microsoft Dynamics AX web part in Microsoft Dynamics AX 2012. For example, the Business Overview web part provides Microsoft Dynamics AX user context awareness that is lacking in the generic KPI List web part. KPIs are filtered based on the context of the Microsoft Dynamics AX company and partition (for Microsoft Dynamics AX 2012 R2) when they are shown in Role Centers. Also, when a user changes the Microsoft Dynamics AX language to German, for example, the Business Overview web part can switch the labels for the KPI to German.

If you are familiar with the Business Overview web part from Microsoft Dynamics AX 2009, you know that it had two modes. In Microsoft Dynamics AX 2012 R2, the functionality of these two modes has been divided into separate web parts: the Business Overview web part and the KPI List web part. You no longer have to switch modes. If you want to display KPIs, use the KPI List web part. If you want to display indicators, use the Business Overview web part. The two web parts appear in the SharePoint Web Part gallery), as shown in Figure 10-35.

FIGURE 10-35 Microsoft Dynamics AX web parts in the SharePoint Web Part gallery.

Both the Business Overview web part and the KPI List web part have some additional features:

- You can define multiple filters when displaying a KPI or an indicator. In Microsoft Dynamics AX 2012, you could add a relative time filter only to a KPI displayed in the Business Overview web part.
- You can add a Microsoft Dynamics AX menu item or a URL as a drill-through target to a KPI.
- You can limit the number of values that are displayed on the screen when splitting a KPI with a given value.
- They provide better error handling and graceful exit in case of errors that are caused by cube configuration issues.
- The Business Overview web part is extensible. You can create a custom skin for the Business Overview web part and extend its functionality to suit your own business area.

Add a KPI to the KPI List web part

Start the Microsoft Dynamics AX client and then navigate to a Role Center. Note that the example in this section uses a sample Role Center. Select the option to edit the Role Center page. If you are using the Microsoft Dynamics AX Windows client, you are able to personalize the page for yourself only. If you are a developer, customizing the page for everyone, you should start the Microsoft Dynamics AX Enterprise Portal web client and edit the page.

Select Add Web Part. You should see the SharePoint Web Part gallery, as shown in Figure 10-36.

Select the KPI List web part. Click Add. After the web part is added, select Exit Editing. Now you will see the new web part added to the Role Center page.

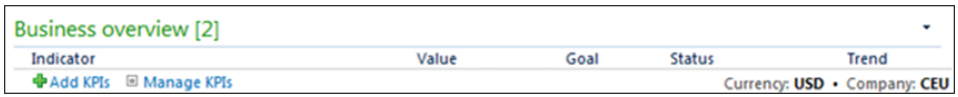


FIGURE 10-36 Adding a KPI.

Click the Add KPIs option to add a new KPI to the web part. You will see an Add KPI dialog box similar to the one in Figure 10-37.

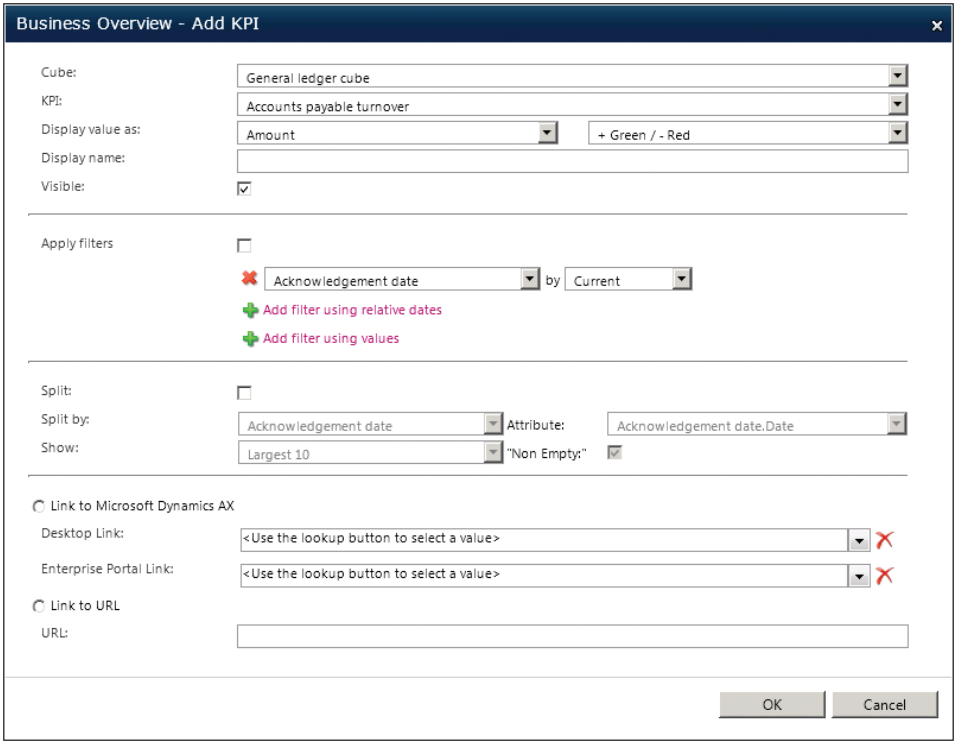


FIGURE 10-37 The Business Overview–Add KPI dialog box.

If you are familiar with the Business Overview web part in Microsoft Dynamics AX 2009 or Microsoft Dynamics AX 2012, you will notice several new additions to the Business Overview – Add KPI dialog box (the Add New Indicator dialog box provides similar options).

First, you have an expanded set of options for applying filters. You can add any number of filters—both relative time periods and fixed values. This way, a user can add a filter to an existing KPI definition and display it on his or her Role Center. This feature yields two benefits. You can define a general-purpose KPI definition that applies to the entire organization or the business unit. Users can narrow down the scope of the KPI definition so that it closely matches their area of focus, without developer intervention.

You are probably familiar with the Split option that lets a user display the breakdown of a KPI definition by a selected attribute. For example, the Revenue KPI can be split by sales units so that a sales manager can monitor units that are falling behind. Unlike in Microsoft Dynamics AX 2012, the user can display the top 10 or bottom 10 values, so that the list is not too long.

It was possible to provide a drill-through link to each KPI in Microsoft Dynamics AX 2012, but the picking experience was not user friendly. In Microsoft Dynamics AX 2012 R2, the picking experience has been improved so that the user can associate a Microsoft Dynamics AX menu item or a URL with each KPI.

Notice that the *Cube* field is already set to a prebuilt cube. In Microsoft Dynamics AX 2012, the Business Overview web part is hardwired to display KPIs from the default cube database. If you want to point the Business Overview web part to a different database, you can specify the database by providing a database connection file; that is, an Open Database Connectivity (ODC) file. For information about how to define an ODC file and add ODC files to Enterprise Portal, see "How to: Create an ODC file for a Business Overview Web Part" at *http://msdn.microsoft.com/en-us/library/hh128831.aspx*.

The default database is specified in the System Administration > Setup > Business Intelligence > Analysis Services > Analysis Servers form. When you deploy an SSAS project by using the SQL Server Analysis Services Project Wizard, the OLAP database created by this action is added to the list of databases in the Analysis Servers form.

Select an analysis server, and then click the OLAP Databases tab (see Figure 10-38).

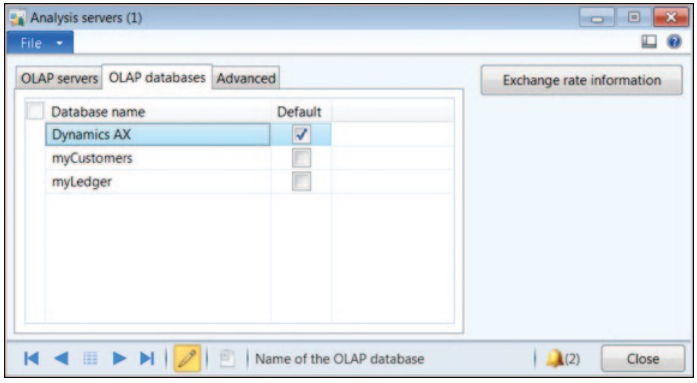


FIGURE 10-38 Analysis Servers form specifying the default OLAP database.

The Default check box specifies the default OLAP database used by the Business Overview web part. You can switch the default database by selecting the check box for a different database.

Add a custom time period filter

Relative time period filters are shown in the Business Overview web part when you add a KPI or an indicator. However, you can define your own time period filter by using the Time Periods form (System Administration > Setup > Business Intelligence > Analysis Services > Time Periods), as shown in Figure 10-39.

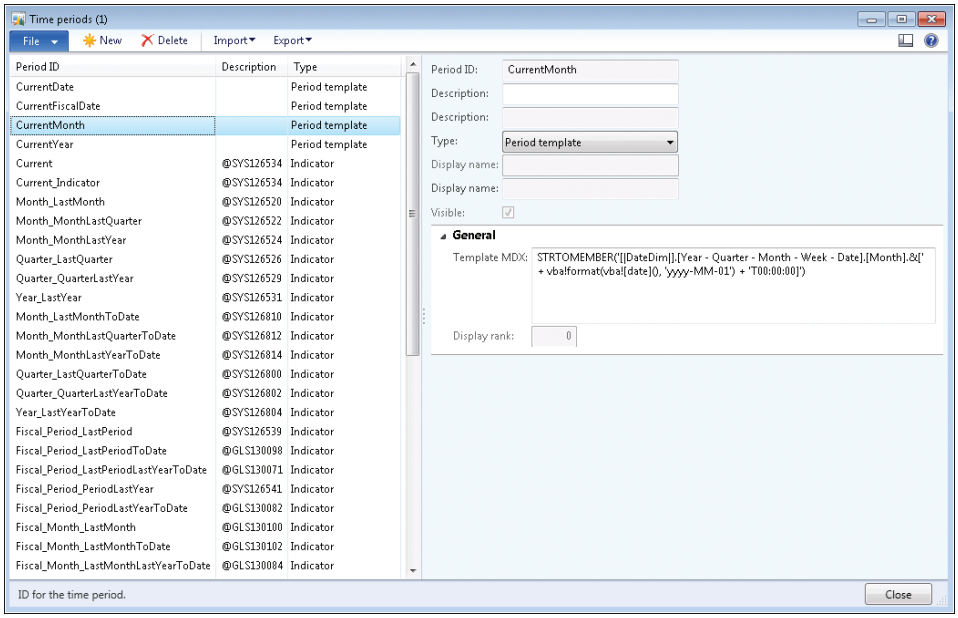


FIGURE 10-39 The Time Period form.

This form lists three types of time periods:

- **Indicators** These define the relative time periods that apply to indicators—the items that you add to the Business Overview web part.
- **KPI lists** These define the relative time periods that apply to KPIs—the items that you add to the KPI List web part.
- **Period templates** These are reusable macros that can be used by both indicator and KPI list entries. Period templates save you from having to recode commonly used patterns repeatedly.

You can define additional indicator and KPI list periods by using MDX code in this form. The Business Overview web part makes these filters available to users at run time.

The following are example definitions to help you understand time period filters.

Period template: CurrentDate

If the time period definition is a template, you need to modify only the MDX expression in the template.

The CurrentDate period template contains the following MDX expression, which gets the current date from the system:

```
STRTOMEMBER('[|DateDim|].[Year - Quarter - Month - Week - Date].[Month].&[' +
vba!format(vba![date](), 'yyyy-MM-01') + 'T00:00:00]')
```

Notice the token |*DateDim*| in the expression. The Business Overview web part replaces this token with the actual name of the date dimension; therefore, you can use this expression with any date dimension.

If you examine the period template definition for CurrentFiscalDate, you will notice another token:

```
STRTOMEMBER('[|FiscalDateDim|].[Year quarter period month date].[Date].&[|c|]&[' +
vba!format(vba![date](), 'yyyy-MM-dd') + 'T00:00:00]')
```

In this case, the system interprets the token |*FiscalDateDim*| as a fiscal date dimension. The system identifies a fiscal date dimension by the name given to the dimension. The system interprets the token |*c*| as the current company.

Indicator: Month_LastMonth

The definition for the Month_LastMonth indicator uses the template that was discussed in the previous section.

The definition for an indicator contains two MDX expressions that correspond to two time period definitions (see Figure 10-40). The expression that provides the value for the current period is defined in the *Current Period* MDX field. Because there is already a template for calculating the current month, you can use that definition by referencing the template *%CurrentMember%*.

The expression that provides the value for the previous period is defined in the *Previous Period* MDX field. Again, you can use the template already defined and define an expression by using that template.

You will also need to provide a description and a display name for the period definition, as shown in the left pane of the Time Period form. These descriptions and display names appear in the Business Overview web part when the user applies the period filter.

Develop reports with Report Builder

Report Builder is a report development tool that was created with the user in mind. (By contrast, Visual Studio tools for creating reports focus on the developer.) Report Builder features an Office-like ribbon that is familiar to users.

Report Builder 3.0, which was released around the same time as SQL Server 2008 R2, requires SQL Server 2008 R2 or a later version. A new version of Report Builder is included with SQL Server 2012. For an overview of the capabilities of Report Builder, see "Getting Started with Report Builder" at *http://technet.microsoft.com/en-us/library/dd220460(SQL.110).aspx*.

Earlier versions of Microsoft Dynamics AX provided the capability to generate report models (.smdl files) that could be used to generate reports with Report Builder 1.0. These .smdl models were based on a set of views, called *secure views*, that were generated on top of the Microsoft Dynamics AX OLTP database.

Microsoft Dynamics AX 2012 no longer generates report models for ad-hoc reporting with Report Builder because Report Builder provides excellent capabilities for creating reports with prebuilt cubes. Also, Microsoft Dynamics AX 2012 cubes provide a good source of aggregate data. For step-by-step instructions about how to use Report Builder with OLAP data, see "Create a report by using SQL Server Report Builder to connect to a cube" at *http://msdn.microsoft.com/en-us/library/gg731902.aspx*.

Develop analytic reports by using Visual Studio tools for Microsoft Dynamics AX

Reports developed by using Report Builder are ideal for scenarios in which users require the capability to create reports for their own consumption or for sharing within a group. However, if you want to create an analytic report for broader consumption within the entire organization, you may want to consider using Visual Studio tools.

Reports created with Report Builder have the following drawbacks when used across the organization:

- They are developed in only one language. These reports cannot use Microsoft Dynamics AX labels, and they cannot be rendered in other languages.
- They do not react to the Microsoft Dynamics AX security model.
- They lack debugging capabilities.
- They mix datasets from multiple data sources, such as Report Data Providers (RDPs).

Most of the Role Center reports that extract aggregate data are sourced with analytic datasets.

Developing an analytic report is no different from developing a standard Microsoft Dynamics AX report by using Visual Studio tools. You define a report dataset and then create a report design

to consume the data. For more information about creating a report, see Chapter 9, "Reporting in Microsoft Dynamics AX," and "Walkthrough: Displaying Cube Data in a Report" at *http://msdn.microsoft.com/en-us/library/dd252605.aspx*.

The remainder of this section examines the salient features of an existing report that consumes analytic data. If you open the AR Administrator Role Center, you will notice the Top Customers by YTD sales report. Start Visual Studio 2010 (the Microsoft Dynamics AX VS reporting tools must be installed).

1. In Microsoft Dynamics AX Application Explorer, right-click the CustTopCustomersbyYTDSales report, and then click Edit.
2. Expand the *Data Sets* node, and then expand the TopCustomersYTDSales dataset.

The report model opens, as shown in Figure 10-40.

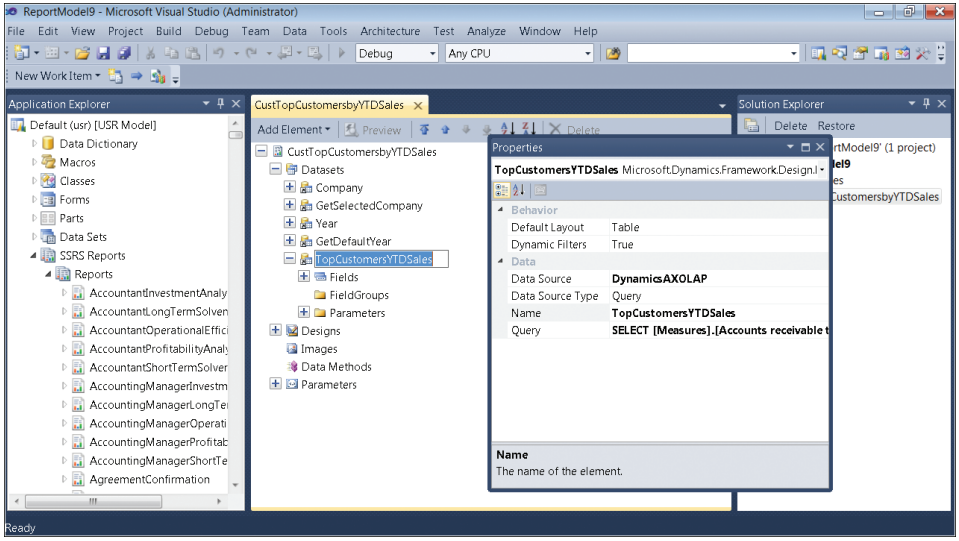


FIGURE 10-40 A report model.

The *Query* property displays the MDX query that was used to retrieve the data. You can click the ellipsis button to open a window where you can modify the MDX query. You can also execute the MDX query from this dialog box (see Figure 10-41).

Note When you create an analytic report, unless you are an MDX expert, you will probably want to develop the MDX query by using an MDX editor, and then paste it into the Query dialog box.

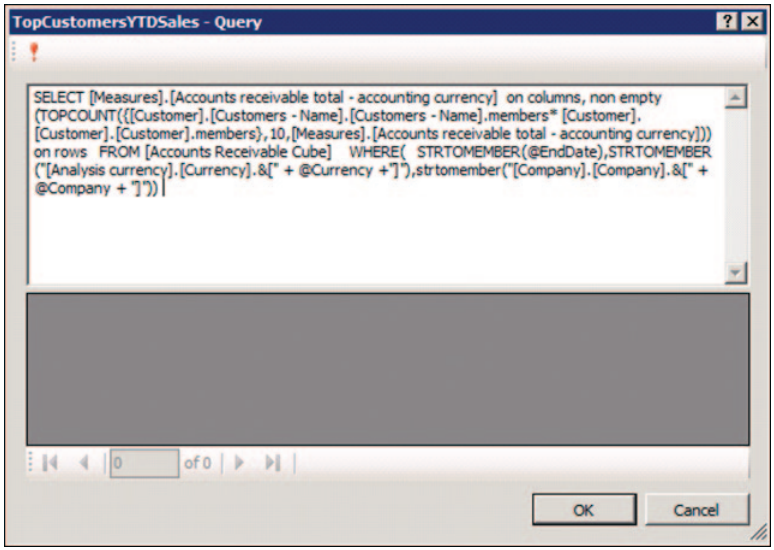


FIGURE 10-41 MDX query dialog box.

Notice that the data source is *DynamicsAXOLAP*, which indicates that the data is sourced from the prebuilt BI solution. To find out which database the data source points to, examine the properties of the *Report Datasources* node in the AOT, as shown in Figure 10-42.

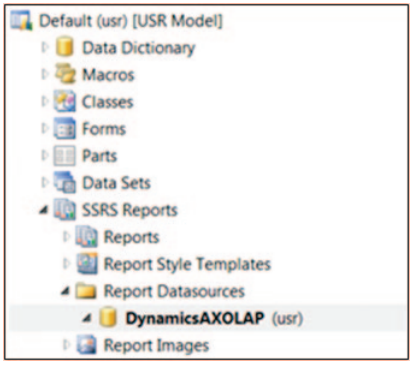


FIGURE 10-42 *Report Datasources* node in the AOT.

DynamicsAXOLAP points to the default cubes. This data source is deployed to the SSRS server as a report data source when the report is deployed. If the report was deployed from a development environment, the report points to the development instance of cubes. If the report was deployed from a test instance, it points to the corresponding cube instance.

To examine the properties of the data connection that is deployed to SSRS, locate the connection file DynamicsAXOLAP in SSRS Report Manager, and then open the file. You will see details about the data connection, as shown in Figure 10-43. In a Microsoft Dynamics AX 2012 R2 environment with multiple partitions, the framework resolves the connections at run time.

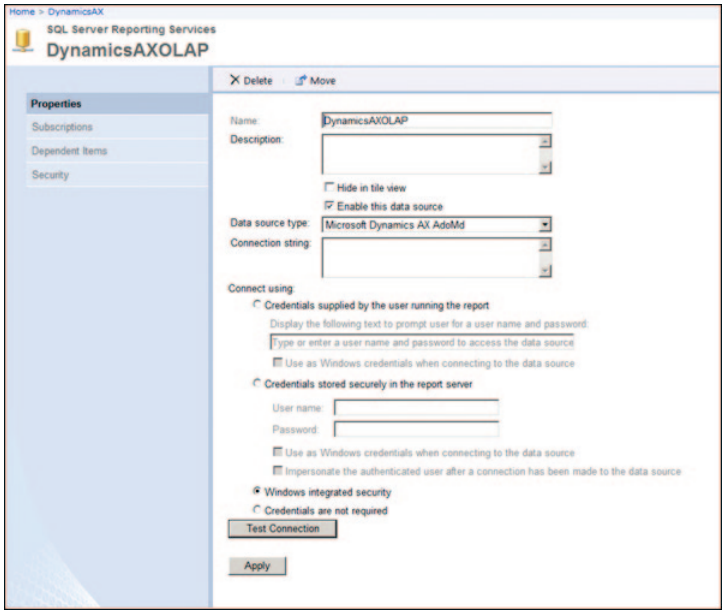


FIGURE 10-43 The *DynamicsAXOLAP* data connection.

Notice that Microsoft Dynamics AX has its own data extension to access Microsoft Dynamics AX cubes that are included with Microsoft Dynamics AX 2012.

The section "Add a KPI to the Business Overview web part" earlier in this chapter described how to switch the OLAP database so that the Business Overview web part points to a non-default OLAP database. In that case, you were able to change the SSAS server and the database that were designated as the default server. One important point to remember is that changing the default SSAS database in the Analysis Servers form does not automatically change the default destination of the DynamicsAXOLAP data source that is used for reports.

You can change the data connection by using the following Windows PowerShell command:

```
Set-AXReportDataSource -DataSourceName DynamicsAXOLAP -ConnectionString
"Provider=MSOLAP.4;Integrated Security=SSPI;Persist Security Info=True;Data
Source=[SSASServerName];Initial Catalog=[DatabaseName]"
```

You can also change the connection string in the data connection deployed to the SSRS server by modifying the properties. However, keep in mind that each time you deploy a report, it will be overwritten with the data source connection in the AOT.

If you want to create analytic reports that point to a non-default cube database (for example, a cube database that you create by using the OLAP framework), you must create your own report data source in the AOT. You can use the same Windows PowerShell command that you use to change the data connection. In this case, however, you should provide a new data source name. For more information, see "Set-AXReportDataSource" at *http://technet.microsoft.com/EN-US/library/hh580547*.

CHAPTER 11

Security, licensing, and configuration

In this chapter

Introduction 351
Security framework overview 351
Develop security artifacts 356
Validate security artifacts 363
Create extensible data security policies 364
Security coding 369
Licensing and configuration 376

Introduction

Microsoft Dynamics AX 2012 introduces a new security framework that is based on a model of role-based security. This framework is designed to make maintaining security easier as the security needs of organizations evolve. It also simplifies the process of implementing base-level security.

System administrators and developers each manage parts of the new security system. Developers create and define the security artifacts that provide access to securable objects. System administrators manage security for users on an ongoing basis.

This chapter describes how the Microsoft Dynamics AX run time implements security, licensing, and configuration, and explains how they determine the portions of interface that the user sees and the data that the user can access. You can use the security framework to create security artifacts that control access to forms, reports, menus, and menu items. Microsoft Dynamics AX 2012 also introduces a new extensible data security framework that lets you restrict access to sensitive data at a granular level so that users see only the data they need to perform their jobs. The licensing and configuration frameworks give you the option to license application modules, thus providing access to various application areas. You can also enable and disable functionality independently of licensing by using configuration keys.

Security framework overview

The Microsoft Dynamics AX security framework consists of three layers: authentication, authorization, and data security. Figure 11-1 provides a high-level overview of the security architecture of Microsoft Dynamics AX. The following sections describe each layer in detail.

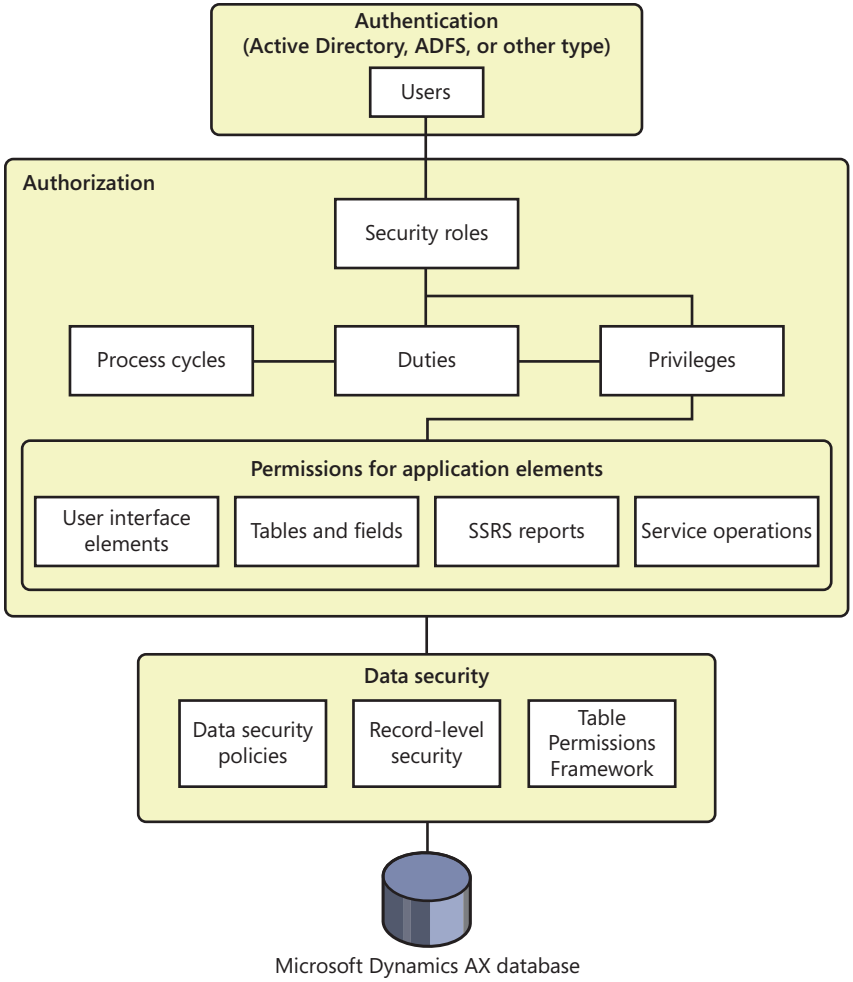


FIGURE 11-1 Microsoft Dynamics AX 2012 security framework.

Authentication

Authentication is the process of establishing the user's identity. Microsoft Dynamics AX users can be authenticated in two ways. The first way is through the use of Integrated Windows Authentication to authenticate Active Directory users. This can be accomplished by either making a specific Windows user a Microsoft Dynamics AX user, or by making an entire Active Directory group a user within Microsoft Dynamics AX. After the Active Directory group is added as a user within Microsoft Dynamics AX, any user who belongs to that Active Directory group can access Microsoft Dynamics AX. The ability to add an Active Directory group as a user within Microsoft Dynamics AX is new for Microsoft Dynamics AX 2012.

The second way of authenticating a user is called *flexible authentication*, which is also new in Microsoft Dynamics AX 2012. With flexible authentication, a user can be authenticated to use the

Microsoft Dynamics AX Enterprise Portal web client without requiring Active Directory credentials. Flexible authentication uses claims-based authentication to verify users in Enterprise Portal. For more information, see the white paper "Flexible Authentication in Microsoft Dynamics AX 2012," at *http://www.microsoft.com/en-us/download/details.aspx?id=29050*.

After a user connects to Microsoft Dynamics AX, the user's authorization within the system is determined. Authorization is discussed in the next section.

Authorization

Authorization, also referred to as *access control*, determines whether a user is permitted to perform a given action. In the Microsoft Dynamics AX application, security permissions are used to control access to individual elements of the application: menus, menu items, action and command buttons, reports, service operations, web URL menu items, web controls, and fields in both the Microsoft Dynamics AX 2012 Windows client and in Enterprise Portal.

In Microsoft Dynamics AX 2012, the new security model follows the principles of role-based access control. This security model is hierarchical; each element in the hierarchy represents a different level of detail, starting with permissions, which are at the bottom:

- *Permissions* represent access to individual securable objects, such as menu items and tables.
- *Privileges* are composed of permissions and represent access to tasks, such as canceling payments or processing deposits.
- *Duties* are composed of privileges and represent parts of a business process, such as maintaining bank transactions.
- *Roles* are composed of duties (and sometimes) privileges that determine a user's access to Microsoft Dynamics AX. These roles correspond to roles within an organization, such as an accountant or human resources manager.

Figure 11-2 shows the elements of role-based security and their relationships.

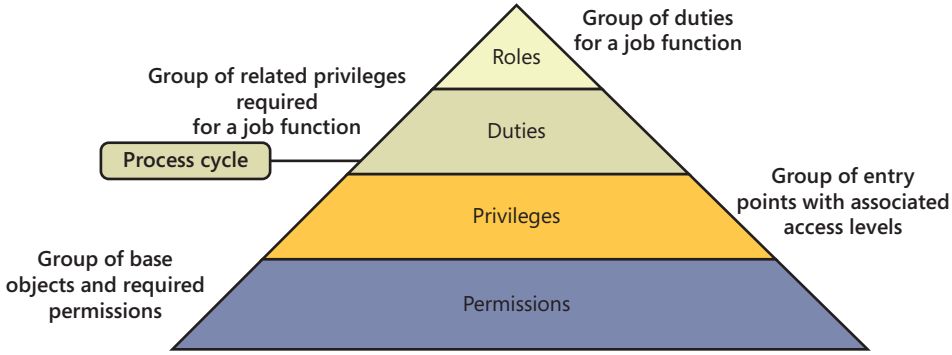


FIGURE 11-2 Elements of role-based security.

The following sections explain the elements of the security model in more detail.

Permissions

In the Microsoft Dynamics AX security model, permissions group together the securable objects and access levels that are required to run a function. These include any tables, fields, forms, or server-side methods that are accessed through an entry point. Menu items, web content items, and service operations are referred to collectively as *entry points*. Each function in Microsoft Dynamics AX, such as a form or a service, is accessed through an entry point.

Only developers can create or modify permissions. The section "Develop security artifacts" later in this chapter explains in detail how to modify permissions.

Privileges

A privilege specifies the level of access that is required to perform a job, solve a problem, or complete an assignment. Privileges can be assigned directly to roles. However, for easier maintenance, it is recommended that only duties be assigned to roles.

A privilege contains permissions to individual application objects, such as user interface elements and tables. For example, the *Cancel payments* privilege contains permissions to the menu items, fields, and tables that are required to cancel payments.

By default, privileges are provided for all features in Microsoft Dynamics AX. A system administrator can modify the permissions that are associated with a privilege or create new privileges.

Duties

A duty is a group of privileges—or tasks—that corresponds to part of a business process. A system administrator assigns duties to security roles. A duty can be assigned to more than one role.

In the security model for Microsoft Dynamics AX, duties contain privileges. For example, the duty *Maintain bank transactions* contains the privileges *Generate deposit slips* and *Cancel payments*. Although both duties and privileges can be assigned to security roles, it is recommended that you use duties to grant access to Microsoft Dynamics AX. By doing so, you can use the segregation of duties functionality explained in the next paragraph.

Security or policies might require that specific tasks be performed by different users. For example, an organization might not want the same person both to acknowledge the receipt of goods and to process payment to the vendor. This concept is called *segregation of duties*. Segregation of duties helps organizations reduce the risk of fraud, and it also helps detect errors or irregularities. By segregating duties, an organization can better comply with regulatory requirements, such as those from Sarbanes-Oxley (SOX), International Financial Reporting Standards (IFRS), and the U.S. Food and Drug Administration (FDA).

In Microsoft Dynamics AX 2012, segregation of duties lets a system administrator specify the duties that should always be segregated and should not overlap for a given user.

Microsoft Dynamics AX includes default duties. However, a system administrator can modify the privileges that are associated with a duty or create new duties. For more information, see the section "Set up segregation of duties rules," later in this chapter.

Process cycles

A business process is a coordinated set of activities in which one or more participants consume, produce, and use economic resources to achieve organizational goals. In the context of the security model, business processes are called *process cycles*. To help the system administrator locate the duties that must be assigned to roles, duties are organized by the business processes that they belong to. For example, in the accounting process cycle, you may find the *Maintain ledgers* and *Maintain bank transactions* duties. Process cycles are used for organization only.

Security roles

Microsoft Dynamics AX 2012 uses role-based access control. In other words, access is not granted to individual users; it is granted only to security roles. The security roles that are assigned to a user determine the duties that the user can perform and the parts of the user interface that the user can view.

Microsoft Dynamics AX 2012 provides the capability to track date-effective data by using valid time state tables. A system administrator can also specify the level of access that the users in a security role have to current, past, and future records on such tables.

By managing access through security roles, system administrators save time because they do not have to manage access separately for each user. Security roles are defined once for all organizations.

A user can be assigned to a security role in several ways. One method is to assign a user to a security role directly. A second method is by assigning an Active Directory group to a role, which assigns all members of the Active Directory group to that role. In addition, users can be assigned to security roles automatically based on business data. For example, a system administrator can set up a rule that associates a human resources position with a security role. Any time that a user is assigned to that position, the user is automatically added to the appropriate security role. This functionality is called *dynamic role assignment*. Typically, a system administrator assigns users to security roles.

Security roles can be organized into a hierarchy so that security roles can be defined as combinations of other security roles. For example, the *Sales manager* security role can be defined as a combination of the *Manager* security role and the *Salesperson* security role. Instead of each security role being defined individually, in a security role, hierarchy security roles can inherit the permissions from other security roles and reuse them.

In the security model for Microsoft Dynamics AX, duties and privileges are used to grant access to the program. For example, the sales manager role can be assigned the *Maintain revenue policies* and *Review sales orders* duties.

By default, sample security roles are provided. All functionality in Microsoft Dynamics AX is associated with at least one sample security role. A system administrator can assign users to the sample security roles, modify the sample security roles to fit the needs of the business, or create new security roles.

Note The sample security roles do not correspond to Role Centers, which are default home pages that provide an overview of information that pertains to a user's work, such as the user's work list, activities, frequently used links, and key business intelligence information.

Data security

As mentioned earlier, Microsoft Dynamics AX 2012 introduces a new extensible data security framework (XDS) that you can use to control access to transactional data by assigning data security policies to security roles. Data security policies can restrict access to data, based either on the effective date or on user data, such as the sales territory or the organization that a user is assigned to.

Note Data security is separate from functional security, which is achieved by using role-based security.

In addition to the XDS, you can use record-level security to limit access to data that is based on a query. However, because the record-level security feature is being deprecated in a future release of Microsoft Dynamics AX, it is recommended that you use the XDS instead.

Additionally, Microsoft Dynamics AX has a table permissions framework to protect data. The table permissions framework allows enforcement of data security for specific tables by the Application Object Server (AOS). Explicit authorization checks are performed when a user tries to access data related to tables protected by the table permissions framework.

Develop security artifacts

Access to a securable object within Microsoft Dynamics AX is controlled through various security artifacts such as permissions, privileges, duties, roles, and policies. You can create and manage these artifacts by using the Application Object Tree (AOT), as shown in Figure 11-3.

Set permissions for a form

You build security from the ground up, beginning at the form level. The first step is to control access to the data in a form. When you save a form in the AOT, Microsoft Dynamics AX automatically discovers all of the tables and other items that the form accesses. This functionality is called *auto-inference*. Auto-inference simplifies configuring table permissions. Based on tables that are used in the form, create, read, update, and delete (CRUD) permissions are set automatically for that form. The system automatically adds or updates the *Read*, *Update*, *Create*, and *Delete* nodes in the AOT under AOT\Forms\<*FormName*>\Permissions.

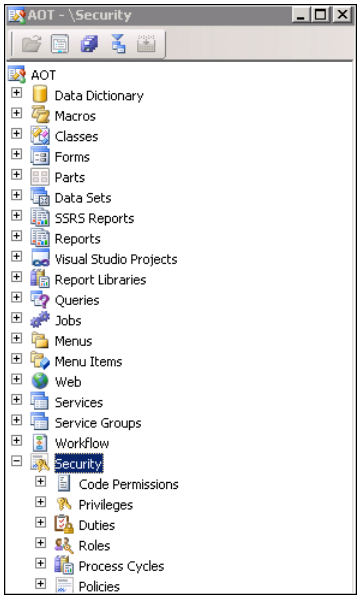


FIGURE 11-3 Security artifacts in the AOT.

Figure 11-4 illustrates the set of permissions for the AgreementClassification form.

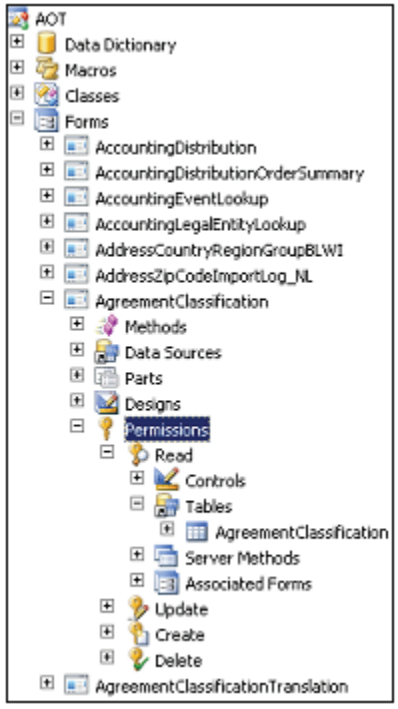


FIGURE 11-4 Read permissions for the AgreementClassification form.

While auto-inference automatically sets the permissions properties for the data sources, you can also set the permissions for a data source manually.

For example, in the Read permissions shown in Figure 11-4, the properties for the AgreementClassification table are set by auto-inference, as shown in Figure 11-5.

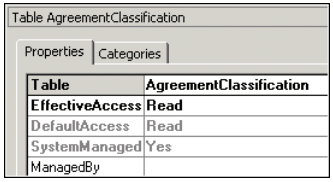


FIGURE 11-5 AgreementClassification table properties set by auto-inference.

The *SystemManaged* property is set to *Yes*. However, you can change the *EffectiveAccess* property to something other than *Read*. In that case, the *SystemManaged* property changes to *No*. This indicates to the security framework that you have chosen to override manually the value set by auto-inference, as shown in Figure 11-6.

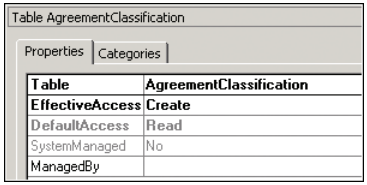


FIGURE 11-6 AgreementClassification table properties set manually.

So far, this section has discussed individual permissions under the *Tables* node. However, you can also set permissions for additional nodes, such as *Controls*, *Server Methods*, and *Associated Forms*.

Note that in the same manner that you set up permissions for forms, you can set permissions to read and write data under the *Permissions* node of several AOT elements, including the following:

- Forms/<*FormName*>
- Parts/Info Parts/<*InfoPartName*>
- Reports/<*ReportName*>
- Web\Web Files\Web Controls\<*WebControlName*>
- Services\<*ServiceName*>\Operations\<*OperationName*>

An *associated form* comes into play when the parent form—in this example, the AgreementClassification form—contains a button that opens another form. In such cases, you should add permissions so that the associated form is accessible to users of the parent form. You can accomplish this by referencing the associated form under the *Associated Forms* node.

When a user has access to a form, by default, the user has access to all of the controls on the form. You can override the default settings by adding permissions nodes for individual controls. You can do this by using the *Controls* node.

Set permissions for server methods

If a server method is tagged with the attribute *SysEntryPointAttribute*, users must have explicit access to that method. If such a server method is invoked through a form, you can control access by adding the method to the *Server Methods* node and explicitly setting the permission to invoke. Any role that provides access to that form through the appropriate permission—in this example, read—also grants permission to the server method.

Set permissions for controls

When you develop a form, Microsoft Dynamics AX provides the ability to add controls to the form as securable objects. These can either be data-bound to the form or unbound. All data-bound controls are configured automatically with security, whereas unbound controls can be managed through code. Security in an unbound control, such as a menu function button, is linked to the referenced object and visibility is controlled through permissions on the referenced object.

Create privileges

After you specify permissions, the next step is to create privileges. As mentioned earlier, a privilege is a set of permissions that provides access to securable objects. By using auto-inferred table permissions and securing menu items with privileges, you control access to the data in a form. The following example (Figure 11-7) links the entry point to the form with the associated permissions.

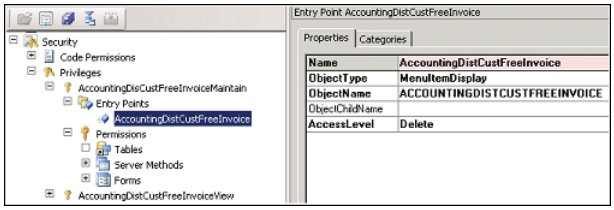


FIGURE 11-7 Linking a form with permissions.

In this example, the privilege *AccountDistCustFreeInvoiceMaintain* contains an entry point, *AccountingDistCustFreeInvoice*. This is a menu item that, in turn, points to a form. Note that in the properties, *AccessLevel* is set to *Delete*. This implies that when a user accesses the form through this particular menu item, the Microsoft Dynamics AX security framework will look under the *Permissions**Delete* node for that form and grant access to the tables that are listed under that node. This example illustrates how the system ties together the privileges, entry points, and permissions and determines the access that the user should have if he or she has access to that privilege through a security role.

A menu item provides an entry point for opening a form. Security properties on the menu item control which sets of form permissions are available to select when privileges are assigned to the menu item.

Each menu item has the following security properties:

- *ReadPermissions*
- *UpdatePermissions*
- *CreatePermissions*
- *CorrectPermissions*
- *DeletePermissions*

These properties refer to the nodes under AOT\Forms\<*FormName*>\Permissions. For example, the *UpdatePermissions* property refers to the node AOT\Forms\<*FormName*>\Permissions\Update.

Table 11-1 describes the values for these permission properties.

TABLE 11-1 Property values for create, update, read, and delete.

Property value	Description
Auto	The default. *Auto* means that the corresponding set of form permissions will be available to select as privileges on this menu item. The privileges will be selected on the privilege node for this menu item, which will be under the *Entry Points* node. The path to the privilege node for this menu item is AOT\Security\Privileges*MyPrivilege*\Entry Points*MyMenuItem*. For example, if the *UpdatePermissions* property is set to *Auto*, the permission set under the node *MyForm*\Permissions\Update will be available to select for privileges under AOT\Security.
No	The opposite of *Auto*. The corresponding permission set will not be available to select as a privilege on the privilege node for the menu item under the *Entry Points* node.

For example, if the *ReadPermisssions* property on a menu item is set to *No*, the menu item will not pick up the *ReadPermissions* property from the form that the menu item references. You can use this method to add a permission to a menu item without affecting the permissions to securable objects that are available through that menu item. This helps restrict the permissions that a system administrator can issue for the menu item when assigning it to a privilege.

In some situations, a menu item points to a class or a report directly. In this case, you would need to link to a class, which itself is not associated with any permissions. In such cases, you need to use a code permission. A *code permission* is a group of permissions that are associated with a menu item or a service operation. If you want to run code directly through a menu item, you must set a code permission for it. Code permissions are also represented as a node within the AOT. When a security role grants access to a menu item, the role also has access to other AOT items that are listed within the code permission for the menu item. The access level is controlled by the permissions that are set under the *Code Permissions* node.

Microsoft Dynamics AX uses the concept of a permission union. If multiple permissions are specified for the same object through multiple privileges and roles, the access on the object is the result of the union of those permissions. For example, if one privilege provides read access to a table and another privilege provides delete access to the same table, and both of them belong to a security role, a user who is assigned to the security role will get delete access to the table.

Assign privileges and duties to security roles

After you generate permissions for the various securable objects, you grant access to those securable objects through security roles. The first step is to create privileges, as described in the previous section. You can then either incorporate these privileges into duties or directly assign them to security roles.

In Figure 11-8, the privilege *AccountingDisCustFreeInvoiceMaintain* contains the entry point *AccountingDisCustFreeInvoiceMaintain*.

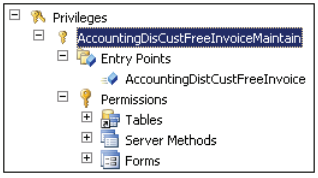


FIGURE 11-8 A privilege containing an entry point.

The entry point is associated with an access level that is specified in the properties (Figure 11-9). Note that in this case, the access level is set to *Delete*. This implies that when the user accesses the entry point, the system will look in the *Permissions\Delete* node for the form that the entry point launches.

Entry Point AccountingDistCustFreeInvoice

Properties | Categories

Name	**AccountingDistCustFreeInvoice**
ObjectType	**MenuItemDisplay**
ObjectName	**ACCOUNTINGDISTCUSTFREEINVOICE**
ObjectChildName	
AccessLevel	**Delete**

FIGURE 11-9 Properties for an entry point.

While it is not mandatory that a privilege be assigned to a security role through a duty, doing so lets the system administrator maintain the privileges through a higher level of abstraction and also lets the system administrator use segregation of duties to meet segregation of duties requirements.

In Figure 11-10, the *CustInvoiceCustomerInvoiceTransMaintain* duty contains the *AccountingDisCustFreeInvoiceMaintain* privilege.

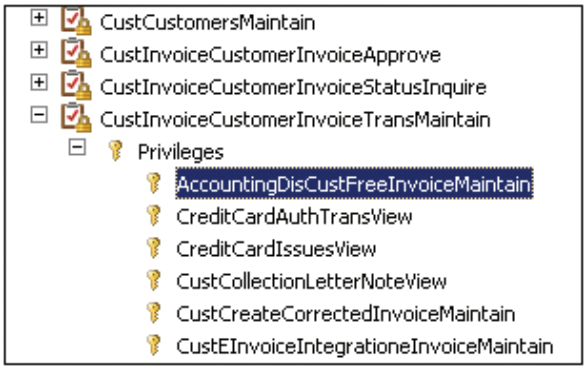


FIGURE 11-10 A duty that contains privileges.

Continuing with the example, notice how the *CustInvoiceCustomerInvoiceTransMaintain* duty is present within the *CustInvoiceAccountsReceivableClerk* role in Figure 11-11.

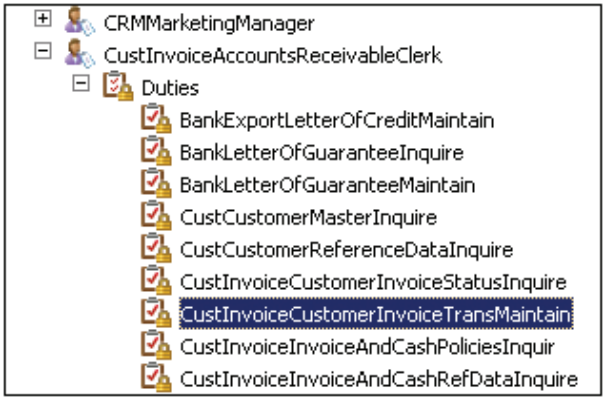


FIGURE 11-11 Duties within a security role.

Note For more information about security, see "Role-based security in the AOT for Developers," at *http://msdn.microsoft.com/en-us/library/gg847971*.

Use valid time state tables

A valid time state table helps you simplify the maintenance of data for which changes must be tracked at different points in time. For example, the interest rate on a loan might be 5 percent for the first year, and 6 percent for the second year. During the second year, you still want to know that the rate was 5 percent during the previous year.

You can set the *ValidTimeStateFieldType* property on a table in the AOT to make the table a valid time state table. Setting this property causes the system to automatically add *ValidFrom* and *ValidTo* columns, which track a date range in each row. The system guarantees that the values in these date or date-time fields remain valid by automatically preventing overlap among date ranges. Data tracked by this type of table is referred to as *date effective*.

Properties on security roles control access to date-effective tables. In the AOT, you can set the properties *PastDataAccess*, *CurrentDataAccess*, and *FutureDataAccess*. By default, these properties are set to *Delete*, which, in effect, means that the tables are not date effective. However, if one of these properties is set to a value other than *Delete*, the property specifies the level of data access for the tables with date-effective fields that are secured by the security role. For example, if a table normally has edit access within the security role and you set the *PastDataAccess* property to *View*, the user can edit current and future data but can only view past data.

Validate security artifacts

After you implement data security, you'll want to make sure that the changes are accurate. The testing process consists of the following steps:

1. Create users.
2. Assign users to roles.
3. Set up segregation of duties rules.

After you complete the steps in this section, start the Microsoft Dynamics AX client as a test user assigned to the appropriate security role (or roles) and ensure that the functional security scenarios work as expected.

Create users

Microsoft Dynamics AX users are either internal employees of your organization or external customers and vendors who require access to Microsoft Dynamics AX for their jobs. Any individual who must access Microsoft Dynamics AX must be added to the list of Microsoft Dynamics AX users in the Users form (System Administration > Common > Users > Users).

Among other options on the form, there is a field called *Account Type*. You must select whether the user or group is authenticated by Active Directory or by a claims-based authentication provider. For Active Directory, the choices are between adding an individual Active Directory user or an Active Directory group as a user.

Assign users to roles

After you create a user within the system, you can assign the user to a security role, either manually or automatically.

You can set up rules for automatic role assignment to guarantee that role membership is based on current business data. If you use automatic role assignment, permissions are updated automatically when people change jobs in an organization. Rules for automatic role assignment run at a fixed interval by using the batch framework. As part of setting up the rule, you specify a query from the list of queries in the AOT to use as a basis for the rule. For more information, see Chapter 18, "The batch framework."

You can assign roles manually when role membership cannot be based on data in Microsoft Dynamics AX. For example, you can assign roles manually if an employee goes on vacation and another employee must perform that employee's duties temporarily. Users who are assigned to security roles manually must also be removed manually by the system administrator. These users are not removed from roles by rules for automatic role assignment.

Set up segregation of duties rules

As mentioned earlier, security or policies might require that specific tasks be performed by different users.

In Microsoft Dynamics AX, when two duties in the same role conflict, or when a user is assigned to two roles that contain conflicting duties, the conflict is logged. You must approve or reject each assignment that causes a conflict. For more information, see "Identify and resolve conflicts in segregation of duties," at *http://technet.microsoft.com/en-us/library/hh556858.aspx*.

Create extensible data security policies

Within any enterprise, some users are restricted from working with certain sensitive data because of confidentiality, legal obligations, or company policy. In Microsoft Dynamics AX 2012, authorization for access to sensitive data is managed through the XDS.

By using the XDS, you can secure data in tables so that users can access only the subset of rows in a table that is allowed by a given policy.

Common uses of extensible data security include the following:

- Allowing sales clerks to see only the accounts they manage.
- Prohibiting financial data from appearing on forms or reports for a specific security role.
- Prohibiting account details or account IDs from appearing on forms or reports for a specific security role.

Extensible data security is an evolution of the record-level security (RLS) that was available in earlier versions of Microsoft Dynamics AX.

Data security policies are enforced on the server tier. This means that extensible data security policies, when deployed, are enforced, regardless of whether data is being accessed through the

Microsoft Dynamics AX client forms, Enterprise Portal webpages, Microsoft SQL Server Reporting Services (SSRS) reports, or .NET services.

Additionally, by using the new framework, you can create data security policies that are based on data that is contained in a different table.

Data security policy concepts

Before developing a data security policy, you need to become familiar with several concepts, such as constrained tables, primary tables, policy queries, and context. This section outlines these concepts. Subsequent sections use these concepts to illustrate how they work together to provide a rich policy framework.

- A *constrained table* is a table (or tables) in a security policy from which data is filtered or secured, based on the associated policy query. For example, in a policy that secures all sales orders based on the customer group, the SalesOrder table would be the constrained table. Constrained tables are always explicitly related to the primary table in a policy.
- A *primary table* is used to secure the content of the related constrained table. For example, in a policy that secures all sales orders based on the customer group, the Customer table would be the primary table. The primary table can also be the constrained table.
- A *policy query* is used to secure the constrained tables specified in an extensible data security policy. This query returns data from a primary table that is then used to secure the contents of the constrained table.
- A *policy context* is a piece of information that controls the circumstances under which a given policy is applicable. If this context is not set, the policy, even if enabled, is not enforced.

  Policy contexts can be of two types: role contexts and application contexts. A *role context* enables the policy to be applied based on the role or roles to which the user is assigned. An *application context* enables a policy to be applied based on information set by the application.

Develop an extensible data security policy

Developing an extensible data security policy involves the following steps:

1. Modeling the query on the primary table.
2. Creating the data security policy artifact in the AOT.
3. Adding the constrained tables and views.
4. Setting the policy context.

Figure 11-12 shows how the *VendProfileAccount* policy is represented within the AOT. Security policies appear under the *Security\Policies* node.

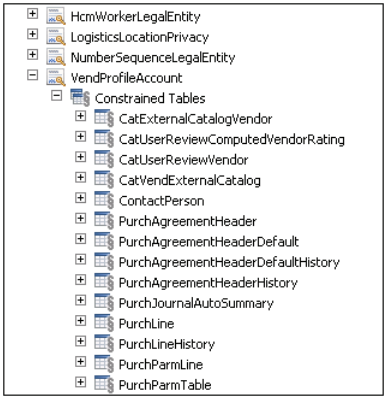


FIGURE 11-12 Security policy in the AOT.

Figure 11-13 shows the properties for this policy.

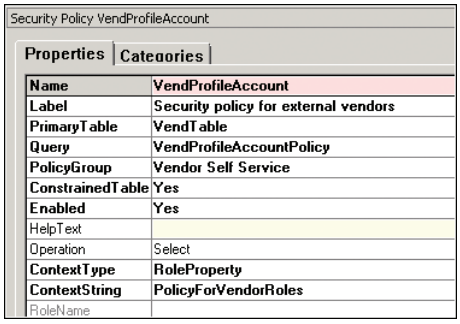


FIGURE 11-13 Properties for a security policy.

Note how the following properties are set on the policy in Figure 11-13:

- The *PrimaryTable* property is set to *VendTable*.
- The *Query* property is set to *VendProfileAccountPolicy*. A policy query is defined in the AOT and can use all of the functionality provided by AOT queries. You model the query with the primary table as the first data source and add more data sources as required. In this example, the additional data sources are defined by the Vendor data model.
- The *Operation* property is set to Select. A policy query could be added to the *WHERE* clause (or *ON* clause) on all *SELECT*, *UPDATE*, *DELETE*, and *INSERT* operations involving the specified constrained tables. In this case, the policy will be enforced only on *SELECT* statements.
- The *PolicyGroup* property is set to *Vendor Self Service*. You use this property to identify groups of related policies. There is no run-time usage of this property.
- The *ConstrainedTable* property is set to *Yes*, which indicates that the primary table is to be secured using this policy. This means that the table from which data is filtered or secured is the same table specified in the *PrimaryTable* property. If this property is set to *No*, the policy is not

enforced on the primary table. You can specify other constrained tables can be specified for the policy, independent of this property.

- The *Enabled* property is set to *Yes*, indicating that the policy will be enforced at runtime.
- The *ContextType* property is set to *RoleProperty*, indicating that the policy is to be applied only if the user is a member of one of a set of roles that have the *ContextString* property set to the same value. In this example, the *ContextString* property value is set to *PolicyForVendorRoles*. If any security roles in the AOT have their *ContextString* property set to *PolicyForVendorRoles*, the policy will be applied if a user belongs to those roles. Besides *RoleProperty*, the *ContextType* property can also be set to *ContextString* or *RoleName*. *ContextString* indicates that you have to specify a value for the *ContextString* property. The security policy uses a specific application context for the policy. The *RoleName* property indicates that the security policy applies only to the user assigned to the value of *RoleName*.

Complex and normalized data models can lead to queries with a large number of joins, which can affect performance. However, in many cases, a significant portion of the policy query retrieves static data, such as the legal entities for the user and the departments to which the user belongs. The XDS provides a way by which this static data can be retrieved less frequently (ranging from once each time the table is accessed to once for each client session) and then reused in subsequent policy applications. This mechanism is called extensible data security constructs.

Extensible data security constructs are tables of type TempDB that are populated according to the *RefreshFrequency* system enumeration value of that table (*PerSession* or *PerInvocation*). They exist in the AOT under the *Data Dictionary\Tables* node.

Figure 11-14 shows an example of an extensible data security construct.

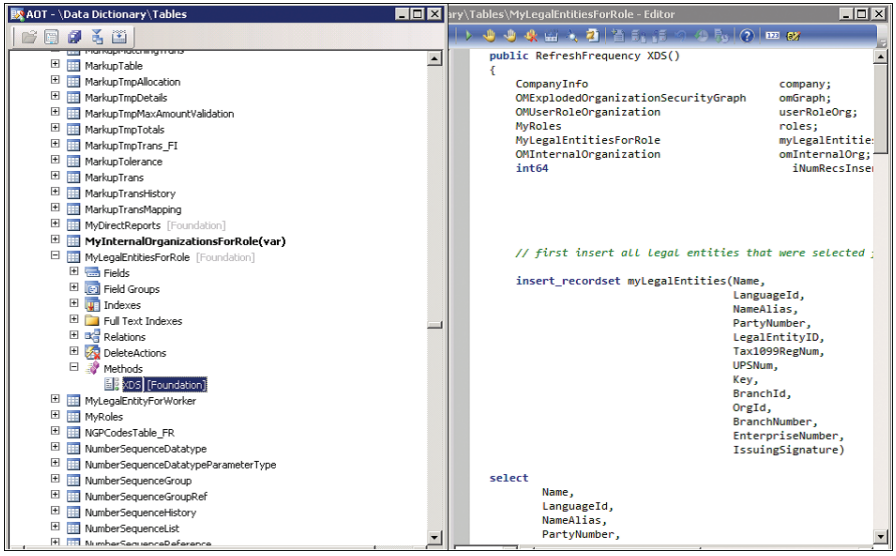


FIGURE 11-14 An extensible data security construct.

The temporary table that is used for the extensible data security construct is written by using a table method named *XDS*. You can use this method to write X++ logic to populate the temporary table. In Figure 11-14, *MyLegalEntitiesForRole* is the extensible data construct that is populated by using the *XDS* method. The logic within the method populates the table with the legal entities that a given user has access to in the context of a role. The *HcmXdsApplicantLegalEntity* query is an example of a policy query that uses the *MyLegalEntitiesForRole* construct. The *HcmXdsApplicantLegalEntity* query involves joins among four data sources. The fourth data source is the *MyLegalEntitiesforRole* construct, which encapsulates several joins. If this *XDS* method sets the frequency to *PerSession*, this TempDB table will be populated the first time this table is referenced in any query at run time. If an extensible data security construct were not used, this query would have involved joins across four more tables on every policy application—a significant performance overhead.

In this scenario, using an extensible data security construct converts a policy query with seven or more joins into a policy query with four joins—a significant performance gain.

Debug extensible data security policies

One of the common issues reported when a customer deploys a new extensible data security policy is that an unexpected number of rows are returned from a constrained table.

The XDS provides a method for debugging problems such as this. The X++ *select* query has been extended with the *generateonly* command, which instructs the underlying data access framework to generate the SQL query without actually executing it. You can retrieve the generated query by using simple method calls.

The following job runs a *select* query on the SalesTable table with a *generateonly* command. It then calls the *getSQLStatement* method on the SalesTable table and generates the output by using the *info* method.

```
static void VerifySalesQuery(Args _args)
{
SalesTable salesTable;
XDSServices xdsServices = new XDSServices();
xdsServices.setXDSContext(1, '');
//Only generate SQL statement for custGroup table
select generateonly forceLiterals CustAccount, DeliveryDate from salesTable;
//Print SQL statement to infolog
info(salesTable.getSQLStatement());
xdsServices.setXDSContext(2, '');
}
```

The XDS further eases the process of advanced debugging by storing the query in a human-readable form. This query and others on a constrained table in a policy can be retrieved by using the following Transact-SQL (T-SQL) query on the database in the development environment (AXBDEV in this example):

```
SELECT [PRIMARYTABLEAOTNAME], [QUERYOBJECTAOTNAME],
[CONSTRAINEDTABLE], [MODELEDQUERYDEBUGINFO],
[CONTEXTTYPE],[CONTEXTSTRING],
```

```
[ISENABLED], [ISMODELED]
FROM [AXDBDEV].[dbo].[ModelSecPolRuntimeEx]
```

The query results are shown in Figure 11-15.

	QUERYOBJECTAOTNAME	CONSTRAINEDTABLE	MODELEDQUERYDEBUGINFO
1	VendProfileAccountPolicy	AssetBook	SELECT * FROM AssetBook(AssetBook_1) EXISTS JOIN 'x' FROM VendTable(VendTabl...
2	VendProfileAccountPolicy	AssetBookMerge	SELECT * FROM AssetBookMerge(AssetBookMerge_1) EXISTS JOIN 'x' FROM VendTa...
3	VendProfileAccountPolicy	AssetDepBook	SELECT * FROM AssetDepBook(AssetDepBook_1) EXISTS JOIN 'x' FROM VendTable(V...
4	VendProfileAccountPolicy	AssetTable	SELECT * FROM AssetTable(AssetTable_1) EXISTS JOIN 'x' FROM VendTable(VendTa...
5	VendProfileAccountPolicy	BankCentralBankPurpose	SELECT * FROM BankCentralBankPurpose(BankCentralBankPurpose_1) EXISTS JOIN '...
6	VendProfileAccountPolicy	BankChequeReprints	SELECT * FROM BankChequeReprints(BankChequeReprints_1) EXISTS JOIN 'x' FROM ...
7	VendProfileAccountPolicy	BankChequeTable	SELECT * FROM BankChequeTable(BankChequeTable_1) EXISTS JOIN 'x' FROM Vend...

FIGURE 11-15 Results from a query on a constrained table.

As you can see in Figure 11-15, the query that will be appended to the *WHERE* clause of any query to the AssetBook table is available for debugging. Other metadata, such as *LayerId*, is also available if needed.

When developing policies, keep the following principles in mind:

- Follow standard best practices of developing efficient queries. For example, create indexes on join conditions.
- Reduce the number of joins in the query. Complex and normalized data models can lead to queries with a large number of joins. Consider changing the data model or adopting patterns such as extensible data security constructs to reduce the number of joins at run time.

Note that when multiple policies apply to a table, the results of the policies are concatenated with *AND* operators.

Security coding

This section covers the Trustworthy Computing features of Microsoft Dynamics AX, focusing on how they affect security coding. This section describes the table permissions framework, code access security (CAS), and the best practice rules for ensuring that the code avoids a few common pitfalls related to security.

Table permissions framework

The table permissions framework provides security for tables that are located in the database and available through the AOT. The *AOSAuthorization* property on a table (Figure 11-16) specifies the operations that must undergo authorization checks when a given user accesses the table.

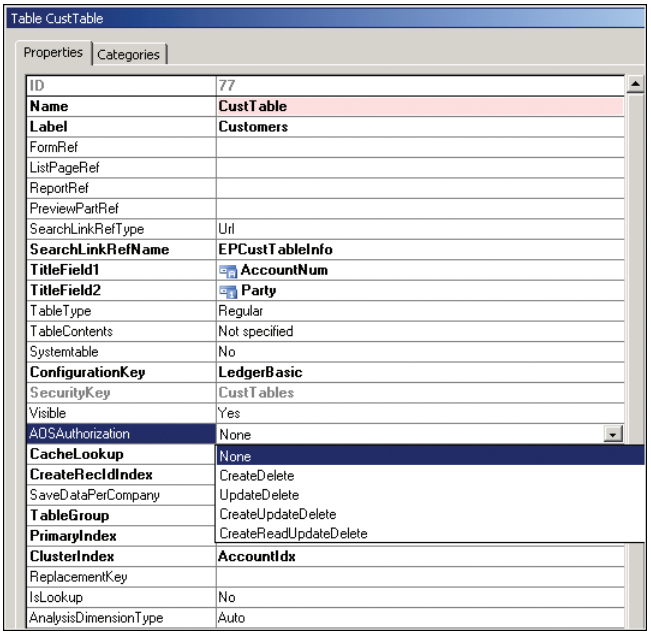


FIGURE 11-16 The property sheet for a table.

The *AOSAuthorization* property is an enumeration. Table 11-2 lists its possible values.

TABLE 11-2 *AOSAuthorization* property values.

Value	Description
None	No AOS authorization validation is performed (default value).
CreateDelete	Create and delete authorization validation is performed on the AOS.
UpdateDelete	Update and delete authorization validation is performed on the AOS.
CreateUpdateDelete	Create, update, and delete authorization validation is performed on the AOS.
CreateReadUpdateDelete	All operations are validated on the AOS.

In addition to the *AOSAuthorization* property, you can add additional rules for validation by using the following table methods:

- *aosValidateDelete*
- *aosValidateInsert*
- *aosValidateRead*
- *aosValidateUpdate*

Note These methods affect performance. All database operations are downgraded to row-by-row when these methods are used.

Microsoft Dynamics AX 2012 also introduces a new class for authorization checks. Use the *SysEntryPointAttribute* class to indicate which authorization checks are performed for a method that is called on the server. When you use this attribute to decorate a method, an authorization check occurs when the class method executes on the server tier.

Additionally, you can add further checking on the basis of the value used in the constructor of the *SysEntryPointAttribute* class, as described in Table 11-3.

TABLE 11-3 *SysEntryPointAttribute* constructor values.

Setting	Description
[SysEntryPointAttribute(true)]	Indicates that authorization checks are performed on the caller for all tables accessed by the method. The *AOSAuthorization* property does not have to be set on these tables in order for these checks to be performed.
[SysEntryPointAttribute(false)]	Indicates that authorization checks are not performed on any tables that are accessed by the method.

Microsoft Dynamics AX 2012 also provides the capability to do server-side trimming. On tables whose *AOSAuthorization* property is set to *CreateReadUpdateDelete*, the *AOSAuthorization* property on individual fields can be set to *Yes* or *No*. The default value of this property is *No*. A value of *Yes* indicates that authorization checks are performed on read and write operations on the field.

If the *AOSAuthorization* property is set to *Yes* for a field and the user does not have access to the field, the value of the field is not returned to the user. This enforces server-side trimming of the data.

Code access security

The code access security (CAS) framework provides methods that can secure application programming interfaces (APIs) against invocation attempts by untrusted code (code that doesn't originate in the AOT). You can make an API more secure by extending the *CodeAccessPermission* class. A class that is derived from the *CodeAccessPermission* class determines whether code accessing an API is trusted by checking for the appropriate permission.

To secure a class that executes on the server tier, follow these steps:

1. Either derive a class from the *CodeAccessPermission* class, or use one of the following derived classes that are included with Microsoft Dynamics AX and skip to step 6:
    - *ExecutePermission*
    - *FileIOPermission*
    - *InteropPermission*
    - *RunAsPermission*
    - *SkipAOSValidationPermission*

- *SqlDataDictionaryPermission*
- *SqlStatementExecutePermission*
- *SysDatabaseLogPermission*

2. Create a method that returns the class parameters.
3. Create a constructor for all of the class parameters that store permission data.
4. To determine whether the permissions required to invoke the API that you are securing exist, override the *CodeAccessPermission.isSubsetOf* method to compare the derived permission class to *CodeAccessPermission*. The following code example shows how to override the *CodeAccessPermission.isSubsetOf* method to determine whether permissions stored in the current object exist in *_target*:

```
public boolean isSubsetOf(CodeAccessPermission _target)

{

    SysTestCodeAccessPermission sysTarget = _target;

    return this.handle() == _target.handle();

}
```

5. Override the *CodeAccessPermission.copy* method to return a copy of an instance of the class created in step 1. This helps to prevent the class object from being modified and passed to the API being secured.
6. Call the *CodeAccessPermission.demand* method before executing the API functionality that you are securing. The method checks the call stack to determine whether the permission that is required to invoke the API has been granted to the calling code.

When you secure an API by using this procedure, you must call the *assert* method in the derived class prior to invoking the API. Otherwise, the *exception::CodeAccessSecurity* exception is thrown.

Best practice rules

The Best Practices tool can help you validate your application logic and ensure that it complies with the Trustworthy Computing initiatives. The rules that apply to Trustworthy Computing are grouped under General Checks\Trustworthy Computing in the Best Practice Parameters dialog box, as shown in Figure 11-17. The Best Practice Parameters dialog box is accessible from the Development Workspace: On the Tools menu, point to Options > Development, and then click Best Practices.

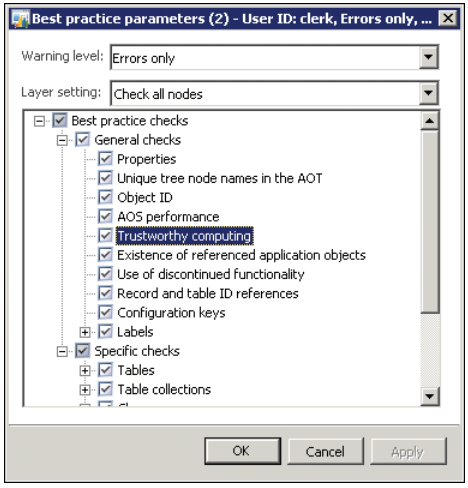


FIGURE 11-17 Best Practice Parameters dialog box with Trustworthy Computing rules.

For more information about the Best Practices tool, see Chapter 2, "The MorphX development environment and tools."

Security debugging

To assist with debugging security constructs, shortcut menus are available in the AOT on some security nodes to help you find objects and roles that are related to a particular security construct. Depending upon where you are looking in the security hierarchy (Figure 11-2), you have the option to view items up or down the hierarchy. For example, for a given duty, you can see all of the roles that the duty belongs to and all of the related privileges and other security objects that are contained within the duty. You can use this information to debug issues related to access levels of various securable objects.

Here is an example of how you can use this feature for a duty:

1. In the AOT, expand Security\Duties.
2. Right-click any duty node, point to Add-Ins > Security Tools, and then click either View Related Security Objects or View Related Security Roles.
3. Examine the rows in the grid control on the form that is displayed. Figure 11-18 shows an example of the form that is displayed when you click View Related Security Objects for a node under AOT\Security\Roles.

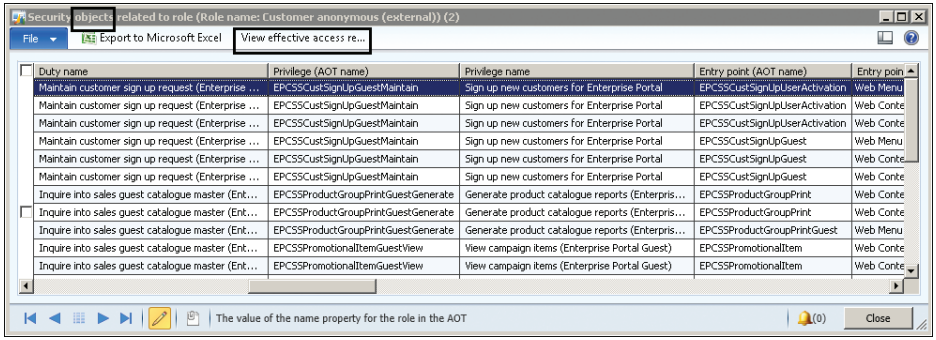


FIGURE 11-18 Security objects for a role.

Note that when you view the related security objects for a role, you also have the option to view the effective access (as highlighted in Figure 11-18) that the role provides to the objects that the role is securing. For example, if the role grants read access to a table through one privilege and delete access through another, the effective access on the table is delete. Therefore, the View Effective Access Results would list that table with delete permissions.

Table 11-4 lists the menu options that are available for various AOT artifacts. The left column of the table lists the nodes that appear in the AOT. The other columns list the menu options.

TABLE 11-4 Menu options for security artifacts.

Artifact name	View Related Security Objects	View Related Security Roles
Security\Role	Available	Not available
Security/Role\<*RoleName*>\Sub Roles	Available	Not available
Security\Duty	Available	Available
Security\Privilege	Available	Available
Data Dictionary\Tables	Not available	Available
Forms	Not available	Available
Menu Items (Display, Output, Action)	Available	Available
Web\Web Menu Items (URLs, Actions)	Available	Available
Web\Web Content\Managed	Available	Available
Data Dictionary\Views	Not available	Available
Parts\Info Parts	Not available	Available
SSRS\Reports\<*Reportname*>\Design	Not available	Available
Web\Web Files\Web controls	Not available	Available
Services\<*ServiceName*>\Operations\<*Anyoperation*>	Available	Available
Security\Code Permissions	Not available	Available

Debug security roles

You can debug standard X++ code in the X++ debugger if you are a member of the *System Administrator* role in Microsoft Dynamics AX. However, you cannot debug issues related to security roles when running Microsoft Dynamics AX as a system administrator because starting the Microsoft Dynamics AX client as a system administrator does not limit the functional security to the security role that you are attempting to debug.

To work through a scenario like this, choose a user who is a member of the *System Administrator* role, assign the role that you want to debug (such as *Accountant*) to that user, and then follow these steps:

1. Close all instances of Microsoft Dynamics AX.
2. Open the Microsoft Dynamics AX Development Workspace.
3. Open another instance of the Microsoft Dynamics AX client.
4. Add the role that you want to test to your Microsoft Dynamics AX user ID:
   - In System Administration, point to Common > Users > Users.
   - Double-click your Microsoft Dynamics AX user ID to open the details page about your account.
   - Assign the security role that you want to test to your Microsoft Dynamics user ID.
5. Close Microsoft Dynamics AX.
6. In the Development Workspace, set breakpoints in the X++ code that you want to debug.
7. Create a job, add the following line, and then execute the job:

```
SecurityUtil::sysAdminMode(false);
```

8. In the Development Workspace, press Ctrl+W to open the application workspace.

You have now opened the client with the permissions of the security role that you want to test and can debug the X++ code.

Note This procedure works for the Microsoft Dynamics AX client, but not for Enterprise Portal or for code executed by using the X++ RunAs API.

To set your environment back to the *System Administrator* role, update the job you created in step 7 with the value set to *true*:

```
SecurityUtil::sysAdminMode(true);
```

By using these steps, you can debug the application while starting it in a mode that simulates its functionality for the role that you want to debug.

Licensing and configuration

Microsoft Dynamics AX 2012 introduces a new licensing model called Named User license. This licensing model provides a simplified way for an organization to license Microsoft Dynamics AX. In Microsoft Dynamics AX 2009, business-ready, module-based, and concurrent user licensing models were available for customers. These licensing models no longer apply to Microsoft Dynamics AX 2012. Instead, the following models have been introduced:

- **Server license** Includes one AOS instance. Additional AOS instances are available by purchasing additional server licenses.
- **User Client Access License (CAL)** Gives a named user access rights to certain capabilities from any number of devices. There are four types of CALs (see the section "Types of CALs" later in this chapter). You can view the user licenses used in the product through a report in System Administration > Reports > Licensing > Named User License Counts.
- **Device CAL** Covers one instance of a device.

Note The intention of this section is to give you a solid overview of the concepts of license keys, configuration keys, and client access license types for development purposes. For more information about pricing and licensing requirements, see the Microsoft Dynamics AX 2012 Licensing Guide at *http://www.microsoft.com/en-us/download/details.aspx?id=29859*.

Even though the software no longer uses module-based licensing, it is still locked with license codes (sometimes referred to as *license keys* or *activation codes*). License codes are used to activate the Microsoft Dynamics AX 2012 software and feature sets that are available in the product. License codes are different from license entitlements (what you are entitled to run and use is based on the Named User licenses that you have acquired). When you acquire a license file for activating the software through Microsoft or a partner, license keys for all feature sets are provided by default. However, the number of users who are allowed to use the product and the type of access that those users are entitled to are based on Named User licenses.

Unlocking a license code is the first step in configuring Microsoft Dynamics AX, because the license code references the configuration key that unlocks a feature set. You can enter the license code by using the License Information form, shown in Figure 11-19, which you access from System Administration > Setup > Licensing > License Information.

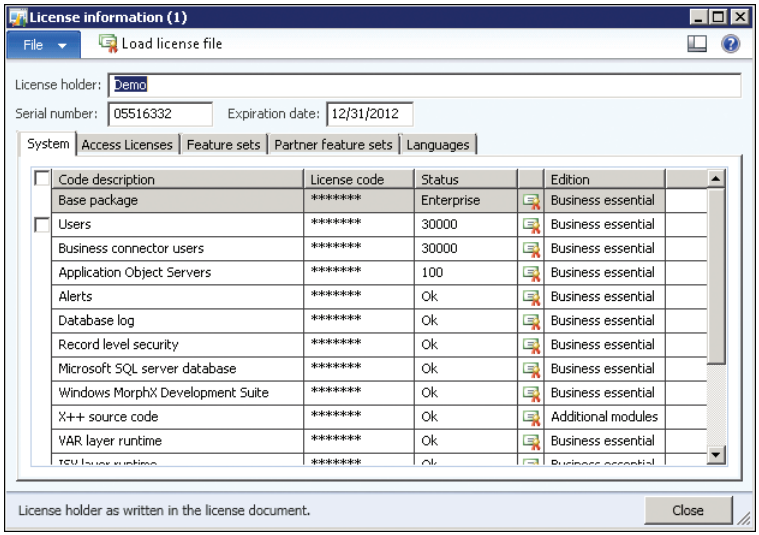


FIGURE 11-19 License Information form.

You enter the license codes manually or import them by clicking Load License File. Microsoft supplies all license codes and license files that are available for a particular release.

License codes are validated individually based on the license holder name, the serial number, the expiration date, and the license code being entered or imported. The validation process either accepts the license code (and updates the status field with counts, names, or OK) or displays an error in the Infolog form.

Note Standard customer licenses do not contain an expiration date. Licenses for other uses, such as evaluation, independent software vendor projects, education, and training, do include an expiration date. When a license reaches its expiration date, the system changes execution mode and becomes a restricted demo product.

License codes are divided into five groups—System, Access Licenses, Feature Sets, Partner Feature Sets, and Languages—each based on the type of functionality it represents, as shown in Figure 11-19. The license codes are created in the AOT and the grouping is determined by a license code property. The Partner Feature Sets tab lets partners include licensed partner modules. The licensing framework can also track dependencies among various license codes. A license code can have up to five prerequisites. Adding a prerequisite for a license code prevents users from removing license codes and disabling feature sets that another feature depends on.

Configuration hierarchy

License codes are at the top of the configuration hierarchy, which is the entry point for working with the configuration system that surrounds all of the application modules and system elements that are available within Microsoft Dynamics AX. The configuration system is based on approximately 300 configuration keys that enable and disable functionality in the application for the entire deployment. Each configuration key controls access to a specific set of functions; when a configuration key is disabled, its functionality is removed automatically from the user interface (note that the database schema is not modified, unlike in Microsoft Dynamics AX 2009). The Microsoft Dynamics AX run time renders presentation controls only for items that are associated with the active configuration key or items that are not associated with any configuration key.

The relationship between license codes, configuration keys, and feature sets is hierarchical. An individual license code not only enables a variety of configuration keys, but it also hides configuration keys and their functions throughout the entire system if the associated license code is not valid or not provided. Hiding configuration keys with unavailable license codes reduces the configuration complexity. For example, if a license code is not entered or not valid in the License Information form, the Configuration form hides configuration keys associated with it and displays only the valid license codes and the configuration keys that depend on them. Figure 11-20 shows a typical configuration hierarchy for implementations.

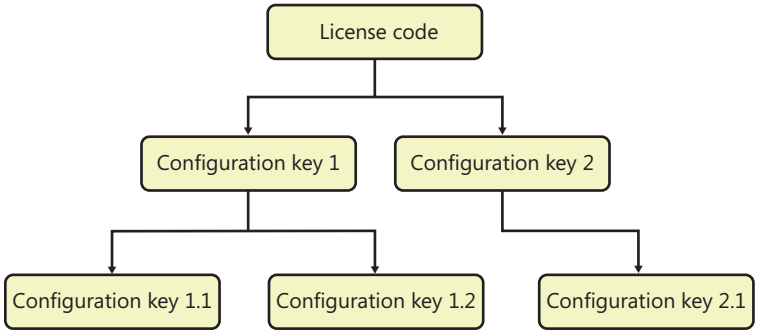


FIGURE 11-20 Configuration hierarchy.

Configuration keys

The application modules and the underlying business logic that license codes and configuration keys enable are available when Microsoft Dynamics AX is deployed. By default, all license codes are enabled; however, only minimal sets of configuration keys are enabled. During setup, system administrators should enable additional configuration keys as required. Within the product, everything from forms, reports, and menus to the Data Dictionary are always present, existing in a temporary state until those feature sets are enabled.

When you enable a configuration key, the feature set associated with that configuration key is enabled. This means that appropriate menu items, submenu items, tables, buttons, and fields are enabled when the configuration key is enabled. A user has access only to those areas that the system administrator has granted access to through security roles and that has been enabled by the configuration key. The parent configuration keys shown in Figure 11-20 are associated with a license code. Removing the license code disables those parent and child configuration keys. If the license code is not disabled, system administrators can enable or disable child configuration keys, thus enabling or disabling the feature sets that they represent.

Note Parent configuration keys can exist without an attached license code. These are available for a system administrator to enable or disable at all times from within the Configuration form (Figure 11-21). However, parent configuration keys that are associated with a license code can only be disabled from the License Information form.

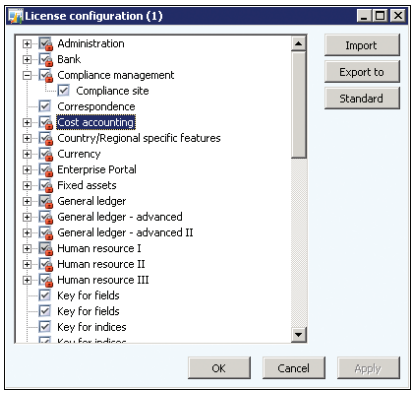


FIGURE 11-21 Configuration form.

As a more detailed example, consider a company that wants most of the functionality in the Trade feature set, but it doesn't do business with other countries/regions. The company, therefore, chooses to not enable the Foreign Trade configuration key, which is a child of the Trade configuration key.

By using the configuration key flowchart shown in Figure 11-22, a system administrator can determine whether a configuration key is enabled, and if not, what it would take to enable it, which depends on the configuration key's parent.

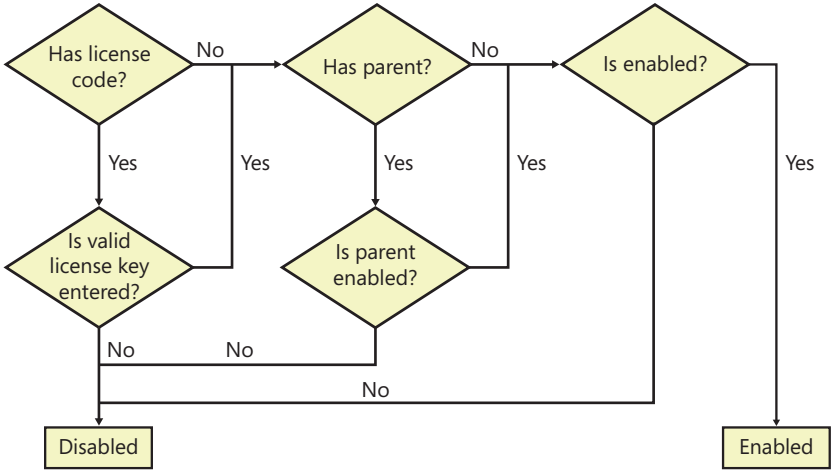


FIGURE 11-22 Configuration key flowchart.

Use configuration keys

An important part of the application development process is mapping extensions to configuration keys that integrate the extensions into the complete solution. Correctly using configuration keys throughout the system can make enterprise-wide deployments flexible and economical, with divisions, regions, or sites all using the same deployment platform and customizing local deployments by using configuration keys rather than by developing specific customizations for each installation. You can't entirely avoid individualized development, however, because of the nature of businesses and their development needs.

Configuration keys affect the Data Dictionary, the presentation, and the navigation infrastructure directly, meaning that you can reference a configuration key property on all relevant elements. Table 11-5 lists the elements that can be directly affected by configuration keys.

TABLE 11-5 Configuration key references.

Grouping	Element types
Data Dictionary	Tables, including fields and indexes Maps Views Extended data types Base enumerations Configuration keys
Windows presentation and navigation	Menus Display: Menu items Output: Menu items Action: Menu items

Grouping	Element types
Web presentation and navigation	URL: Web menu items Action: Web menu items Display: Web content Output: Web content Web menus Weblets
Other	Workflow Approvals Workflow Tasks Workflow Automated Tasks Workflow Types Resources

Types of CALs

The new licensing model, Named User license, provides customers with the ability to use all of the feature sets but provides pricing that is based on the number of users who are using a particular feature instead of pricing that is based on whether a particular module is enabled. In this licensing model, there are four tiers of CALs. Customers are required to comply with Microsoft's licensing terms based on the access rights granted to each user. The following four tiers (user types) are available, listed from highest to lowest level of access (with sample activities):

- **Enterprise** Drives the business and manages processes across the organization
- **Functional** Manages a business cycle within a division or business unit
- **Task** Performs tasks to support a business process or cycle
- **Self-serve** Manages his or her own personal data within the system

All predefined security roles that are included with Microsoft Dynamics AX 2012 belong to one of these four user types, thus giving you the flexibility to license users based on how they are likely to use and derive value from the solution.

The CAL (or user type)-to-security-role mapping is accomplished by first setting the menu item properties *ViewUserLicense* and *MaintainUserLicense* with appropriate user type enumeration. Then, through the security hierarchy, the highest level of user type is evaluated, which essentially becomes the effective user type for the role, as shown in Figure 11-23.

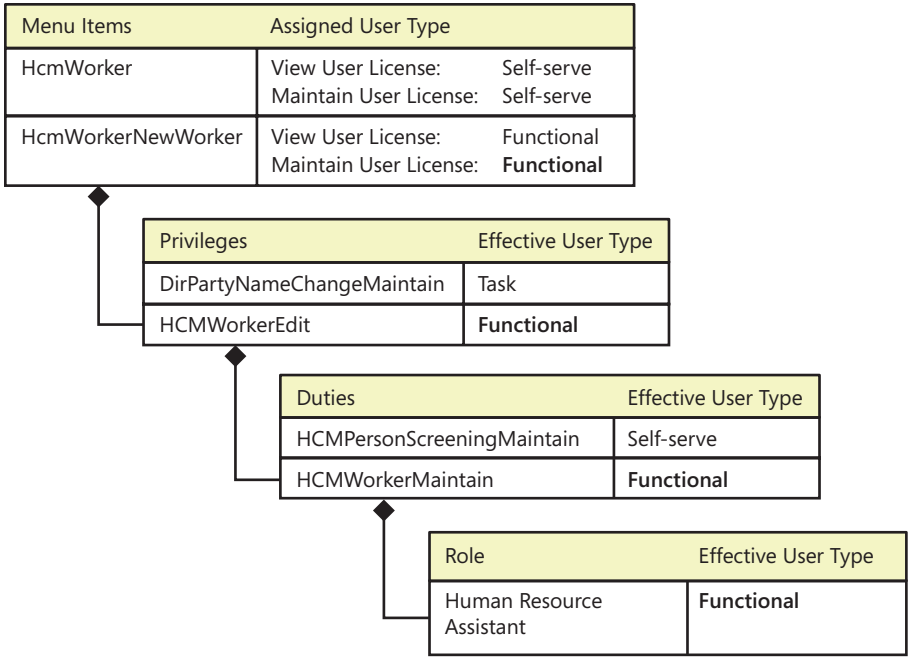


FIGURE 11-23 Security hierarchy and user types.

Note Typically, only Microsoft uses Named User licenses in Microsoft Dynamics AX 2012 to determine the licensing requirements for a customer. This section provides developers with insights into the potential impact that customization might bring to licensing. It is recommended that partners and customers do not modify these values.

As shown in Figure 11-23, Microsoft Dynamics AX 2012 maps a set of menu items to predefined roles by using the security hierarchy. The properties of those menu items are also set with one of the four user-type values. Each user-type value provides the rights to perform actions that only that user type can do. The user type that is required for a given user is determined by the highest level of types among menu items to which that individual has access. For example, to add new workers (access the HcmWorkerNewWorker menu item), the Functional user license is required. Thus, the privilege *HCMWorkerEdit* has an effective user type of Functional, even though it contains the menu item HcmWorker, which is of type Task. Similarly, the highest level of user type flows through the security hierarchy and eventually becomes the effective user type for the role. In this example, the Functional user type is the highest type within the *Human Resource Assistant* role, so the user assigned to the role requires a Functional user license. That user also has license rights to perform actions that are designated to lower user types (such as Task or Self-serve).

Customization and licensing

Given that Microsoft Dynamics AX 2012 uses a security hierarchy and menu items to determine licensing requirements, there are several situations in which customization might affect these requirements.

Changing menu items associated with a role

Each menu item that is included with Microsoft Dynamics AX 2012 is tagged with the appropriate user type. Changing these properties in a higher development layer is intentionally disabled. However, you are free to customize privileges or roles where menu items appear. When a predefined menu item is moved to a different role, that role might require a higher user type. For example, if a menu item tagged with the Enterprise user type is moved into a role that previously only required a Functional user type; the role would require an Enterprise user type going forward. If the menu item is moved into a role requiring an equal or higher user type, there is no impact.

Changing security artifacts associated with a role

Similarly, if privileges, duties, or roles containing menu items with different user types are moved from one security role to another, the user type for the role might be affected. If a privilege that previously had menu items with up to the Functional user type is moved into a role with the Task user type, then the customized role will now require a Functional user type license.

Note When adding new menu items in independent software vendor (ISV) development layers or higher, the system allows you to change the *ViewUserLicense* and *MaintainUserLicense* properties of the menu item. Be aware that specifying license types in custom menu items might affect licensing requirements for customers. It is recommended that customers and partners not assign any license values to these properties. Also, changing menu item properties to a lower user type is intentionally disabled, if the menu item was previously created in the lower development layers.

CHAPTER 12

Microsoft Dynamics AX services and integration

In this chapter

Introduction 385
Types of Microsoft Dynamics AX services 387
Consume Microsoft Dynamics AX services 401
The Microsoft Dynamics AX send framework 411
Consume external web services 414
Performance considerations 415

Introduction

After your company deploys Microsoft Dynamics AX 2012, you can benefit from automating your business processes. But to realize the full potential of Microsoft Dynamics AX 2012 and get the maximum return on investment (ROI) from your Microsoft Dynamics AX 2012 deployment, you should also consider automating interactions between Microsoft Dynamics AX 2012 and the other software in your company and in the companies of your trading partners.

In many business scenarios, external software applications require access to information that is stored in Microsoft Dynamics AX. Figure 12-1 shows a few scenarios in which users access information that is managed in Microsoft Dynamics AX to accomplish a business task. It also shows sample scenarios in which Microsoft Dynamics AX accesses information that is managed in external applications. The arrows indicate the direction in which requests flow.

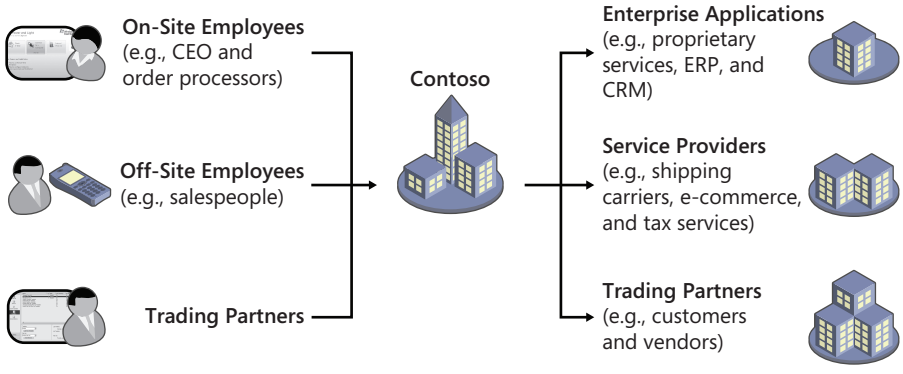


FIGURE 12-1 Common integration scenarios.

If you look at Figure 12-1 you can see that the users on the left side use applications that interact with the Microsoft Dynamics AX data store. These applications send request messages to Microsoft Dynamics AX (for example, to read a sales order). Sometimes, a response is expected from Microsoft Dynamics AX—in this example, the requested sales order document.

In all of these scenarios, another software application exchanges information with Microsoft Dynamics AX to accomplish a task:

- The company's CEO uses an interactive application (such as a Microsoft Office application) to analyze sales data that is stored in Microsoft Dynamics AX. The application communicates with Microsoft Dynamics AX on behalf of the CEO.
- A salesperson who is visiting a prospect's site uses a webpage or a mobile application to create a new customer account and then take the first sales order in Microsoft Dynamics AX from a remote location.
- A sales processor enters a sales order and uses customer records that are stored in a customer relationship management (CRM) application to populate the customer section of the order in Microsoft Dynamics AX.
- Trading partners submit sales orders as electronic documents, which need to be imported into Microsoft Dynamics AX periodically.
- An accountant sends electronic payments or invoices to trading partners.

Performing these tasks manually—without programmatically integrating Microsoft Dynamics AX with other applications and business processes—doesn't scale well and is error prone. With the Microsoft Dynamics AX services framework, you can encapsulate business logic—for example, functionality to create sales orders in Microsoft Dynamics AX—in Microsoft Dynamics AX services. You can then publish these services through the Application Integration Framework (AIF). These Microsoft Dynamics AX services can participate in a *service-oriented architecture* (SOA).

Note SOA is a significant area of software development. A complete discussion of SOA is outside the scope of this book. Good information is available about SOA, including the Organization for the Advancement of Structured Information Standards (OASIS) specification "Reference Model for Service Oriented Architecture 1.0," and the book *Service-Oriented Architecture: Concepts, Technology, and Design* by Thomas Erl.

The Microsoft Dynamics AX service framework provides a toolset for creating, managing, configuring, and publishing Microsoft Dynamics AX services, so that the business logic encapsulated in the service can be easily exposed through service interfaces. All service interfaces that are published through the Microsoft Dynamics AX service framework are compliant with industry standards and are based on core Microsoft technologies, including the software development kit (SDK) for Microsoft Windows Server, Microsoft .NET Framework, Windows Communication Foundation (WCF), and Message Queuing (also known as MSMQ).

In addition to the programming model and tools for implementing services, the Microsoft Dynamics AX service framework also includes the following:

- A set of system services and document services that are included with Microsoft Dynamics AX 2012 and are ready for use.
- A set of features for manipulating inbound and outbound messages, such as support for transformations, value substitutions, and so on.
- An extensible integration framework that supports building new Microsoft Dynamics AX services and publishing them through a set of transport protocols such as Message Queuing, file, HTTP, or Net.tcp.

Note The concept of service references has been removed from Microsoft Dynamics AX 2012.

Publishing Microsoft Dynamics AX services is a simple task that a Microsoft Dynamics AX administrator can do at run time. After a Microsoft Dynamics AX service has been published, external client applications, or *service clients*, can consume it.

Note This chapter discusses configuration and administration tasks only where necessary to help you better understand the development scenarios. For additional details and code samples, see the Microsoft Dynamics AX 2012 system administrator documentation on TechNet (*http://technet.microsoft.com/en-us/library/gg731797.aspx*), or the Microsoft Dynamics AX 2012 Developer Center on MSDN (*http://msdn.microsoft.com/en-us/dynamics/ax/gg712261*).

Types of Microsoft Dynamics AX services

Microsoft Dynamics AX 2012 recognizes three types of services—system services, custom services, and document services—each with its own programming model. Microsoft Dynamics AX publishes metadata about available services and their capabilities in the form of Web Services Description Language (WSDL) files, which can be used for automatic proxy generation.

The following sections explain each type of service in more detail.

System services

Microsoft Dynamics AX system services are generic, infrastructural services that are not tied to specific business logic. System services are included with Microsoft Dynamics AX 2012 and are automatically deployed, so that Microsoft Dynamics AX components and external components can assume that these services are always available.

The functionality published by system services is often used by interactive clients that need to inquire about the capabilities or configuration of a specific deployment at run time. System services and their interfaces are not intended to be modified or reconfigured; they can only be hosted on the Application Object Server (AOS) and cannot be invoked through asynchronous transport mechanisms such as Message Queuing.

Microsoft Dynamics AX system services include the following:

- **Query service** Publishes service operations that allow execution of existing (static) or ad hoc queries from service clients and returns results in the form of generic .NET datasets.
- **Metadata service** Can be used to request information from Microsoft Dynamics AX about its metadata, such as tables, queries, forms, and so on, and thus, about its configuration.
- **User session info service** Can be used to retrieve certain settings for the environment in which requests for the current user are executed; for example, a client application can use the user session service to request information about the current user's currency, company, and time zone, among other things.

Custom services

You can use Microsoft Dynamics AX custom services to publish eligible X++ methods as service operations through integration ports for consumption by external client applications. To do that, you use the programming model for custom services to define metadata that determines the shape of the published service operations and data contracts. Custom services do not have to be tied to Microsoft Dynamics AX queries or tables. For example, you can use a custom service to publish functionality to approve an invoice or to stop a payment.

Note Generally, Microsoft Dynamics AX document services are better suited for implementing services that publish standard operations that operate on queries or tables, such as create, read, update, and delete; these operations are often referred to as CRUD operations.

After you define the service operations and data contracts, you can publish your custom services. Their external interfaces can be configured through the respective system administration forms.

Custom service artifacts

To expose an X++ method as a custom service, you need to create the following artifacts:

- **Service implementation class** A class that implements the business logic and exposes it through X++ methods.
- **Service contract** Service-related metadata (no code). The most important service metadata consists of the service operations that are published to external service applications, and a reference to the X++ service implementation class that implements these service operations.

- **One or more data contracts** X++ classes that represent the complex parameter types used for service operations. Data contracts are not needed for primitive data types.

Service implementation class

A service implementation class contains the code that implements the business logic to publish.

You can use any X++ class as a service implementation class. Service implementation classes don't have to implement any interfaces or extend any super-classes. A class definition for a service implementation class *MyService* could look like this:

```
public class MyService
{
}
```

There are, however, constraints that govern which methods of a service implementation class can be published as service operations. Eligible methods are public methods that use only parameters with data types that can be serialized and deserialized; this includes most primitive data types in addition to valid Microsoft Dynamics AX data contracts. In addition, eligible methods must be declared as service operations in the service contract in the Application Object Tree (AOT).

Note Every method that is intended to be published as a service operation must be annotated with the attribute *SysEntryPointAttribute*, which indicates whether authorization checks are to be performed by the AOS.

The following code shows an example of a method that can be declared as a service operation in the AOT, assuming the X++ type *MyParam* is a valid data contract. (For more information, see the section "Data contracts," later in this chapter.)

```
[SysEntryPointAttribute(true)]
public MyParam HelloWorld(MyParam in)
{
  MyParam out = new MyParam();
  out.intParm(in.intParm() + 1);
  out.strParm("Hello world.\n");
  return out;
}
```

Service contracts

Service contracts define which methods of a service implementation class are publishable as service operations and provide additional metadata that specifies how these methods should be published.

Note Declaring a method as a service operation does not publish that method as a service operation.

To create a new service contract, you need to create a new child node in the AOT under the *Services* node; for example, *MyService*.

The newly created AOT node has a few properties to initialize before any methods of the service can be published as service operations:

- **Service implementation class** This required property links the service interface to the service implementation class. In this example, the value is *MyService*.
- **Namespace** Optionally, you can specify the XML namespace that should be used in the WSDL. If the XML namespace isn't specified, *http://tempuri.org* is used by default. This example uses the namespace *http://schemas.contoso.com/axbook/2012/services*.
- **External name** Optionally, you can assign an external name for each service. In this example, the external name is left blank.

Finally, you need to add service operations to the service contract. To do this, expand the new AOT node, right-click, and then point to Operations > Add Operation.

Note that you can publish only methods as service operations that have been explicitly added to the service contract in the AOT.

Data contracts

A data contract is a complex X++ data type that can be used for input and output parameters in service operations. Most importantly, data contracts must be serializable. You can control how an X++ class is serialized and deserialized by the Microsoft Dynamics AX service framework through the X++ attributes *DataContractAttribute* and *DataMemberAttribute*:

- *DataContractAttribute* declares an X++ class as a data contract.
- *DataMemberAttribute* declares a property as a member of the data contract.

The following code shows a sample definition for the data contract *MyParam*, which is used in the previous example:

```
[DataContractAttribute]
public class MyParam
{
  int intParm;
  str strParm;
}
```

The following code shows a sample property that is included in the data contract:

```
[DataMemberAttribute]
public int intParm(int _intParm = intParm)
{
  intParm = _intParm;
  return intParm;
}
```

X++ collections as data contracts

If you want to use X++ collection types in data contract definitions, you need to ensure that all contained elements are of a data type that is supported for data contracts. Moreover, you need to provide additional metadata with the definition of the service method that uses the parameter, specifying the exact data type of the values in the collection at design time. You do this by using the X++ attribute *AifCollectionTypeAttribute,* as shown here for a sample method *UseIntList()*:

```
[SysEntryPointAttribute(true),
  AifCollectionTypeAttribute('inParm', Types::Integer)]
public void UseIntList(List inParm)
{
  ...
}
```

The two parameters you need to pass into the constructor of the attribute are the name of the parameter to which the metadata is to be applied (*inParm* in the example) and the type of elements in the collection (*Types::Integer* in the example).

If you want to store X++ class types in your collection, you must also specify the class, as shown in the following example:

```
[SysEntryPointAttribute(true),
  AifCollectionTypeAttribute('return', Types::Class, classStr(MyParam))]
public List ReturnMyParamList(int i)
{
  ...
}
```

The three parameters that are passed into the *AifCollectionAttribute* constructor are the name of the parameter (*return*), the type of the elements of the collection type (*Types::Class*), and the specific class type (*MyParam*).

> **Note** The parameter name *return* is reserved for the return value of a method.

Register a custom service

After you create all of the artifacts that are necessary for the custom service, you need to register the new service with the Microsoft Dynamics AX service framework. To register the service (in this example, *MyService*) with AIF, expand the Services node in the AOT, right-click the node you created earlier, and then point to > Add-Ins > Register Service.

As a result of the registration, you can now publish all declared service operations of your service. For more information, see the section "Publish Microsoft Dynamics AX services," later in this chapter.

Document services

The term *document services* stems from the reality that businesses need to exchange business documents, such as sales orders and invoices, with their trading partners. Document services operate on electronic representations of such business documents.

The Microsoft Dynamics AX implementation of these business documents is also referred to as *Axd* documents. Document services are generated from Microsoft Dynamics AX queries. Wizards automate the process of quickly generating and maintaining all necessary artifacts for document services, with a configurable set of well-known service operations, from queries.

By nature, document services provide document-centric application programming interfaces (APIs)—that is, APIs that operate on *Axd* documents. Examples of document-oriented APIs for a sales order service include *create sales order, read sales order, delete sales order*, and so on. Each of these APIs operates on an instance of a sales order document. *Create sales order*, for example, takes a sales order document, persists it in the Microsoft Dynamics AX data store, and returns the sales order identifier for the persisted instance.

Document services are useful in scenarios that require the exchange of coarse-grained business documents, such as sales orders. In these scenarios, exchanged data is transacted and thorough data validation is important, data exchanges are expensive (for example, because enterprise boundaries are crossed), and response times are not critical. Sometimes, responses are not even expected (one-way communication).

The programming model for document services supports customizations to the artifacts that are generated. Microsoft Dynamics AX includes a set of document services that are ready to use. However, you can customize these services to better fit your business needs. The programming model for document services supports the data access layer features that have been introduced with Microsoft Dynamics AX 2012, such as surrogate key expansion, table inheritance, and date-effectivity. In other words, the Microsoft Dynamics AX service framework supports the development of services that use the tables that take advantage of the new functionality.

Document service artifacts

Just like custom services, all document services in Microsoft Dynamics AX 2012 require a service contract, a service implementation, and a data contract. For document services, these artifacts are generated from *Axd* queries; thus, their default implementation follows conventions and looks as follows:

- **Service contract** Service-related metadata (no code) that is stored in the AOT nodes under the *Services* node, such as *SalesSalesOrderService*. The metadata includes the following:
  - Service operations that are available to external service clients.
  - A reference to the X++ service implementation class that implements these service operations.

- **Service implementation** The code that implements the business logic that is to be exposed. For generated document services, the service implementation includes the following key elements:
  - **Service implementation class** An X++ class that derives from *AifDocumentService* and implements the service operations that are published through the service contract. For example, *SalesSalesOrderService* is the service implementation class for the service contract *SalesSalesOrderService*.
  - ***Axd*<Document> class** An X++ class that derives from *AxdBase*. *Axd*<Document> classes coordinate cross-table validation and cross-table defaulting. There is one *Axd*<Document> class for each document service. For example, *AxdSalesOrder* is the *Axd*<Document> class for *SalesSalesOrderService*. The *AxdBase* class, among others, implements code for XML serialization.
  - **Additional artifacts** Optionally, the AIF Document Service Wizard can generate additional artifacts such as *Ax*<Table> classes.

Note In older versions of Microsoft Dynamics AX, an *Ax*<Table> class was generated for each table referenced from a query that was used to generate an *Axd*<Document> class. By default, in Microsoft Dynamics AX 2012, *Axd*<Document> classes use the *Ax*<Table> class *AxCommon* to access tables. The *AxCommon* class provides a default implementation for all *Ax*<Table> class functionality. *Ax*<Table> classes are needed only in advanced scenarios; for example, when a custom value mapping needs to be implemented for a table field.

- **Data object** An X++ class that represents a parameter type and serves as a data contract. The parameter types that the Create New Document Service Wizard generates derive from *AifDocument* and represent business documents. For example, *SalesSalesOrder* is the data object that is created for the *SalesSalesOrderService*.

For a complete list of document service artifacts, see the "Services and Application Integration Framework (AIF)" section of the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/gg731810.aspx*).

The following sections touch on a few selected topics for both *Axd*<Document> and *Ax*<Table> classes. For more information, see the "AIF Document Services" section of the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/bb496530.aspx*).

Axd<Document> classes

Axd<Document> classes (such as *AxdSalesOrder*) extend the X++ class *AxdBase*. Among other things, *Axd*<Document> classes do the following:

- Implement XML serialization for data objects.

- Invoke value mapping.
- Orchestrate cross-table field validation and defaulting.

Axd<Document> classes provide default implementations for XML serialization for all data objects that are used. These classes derive XML schema definitions used for XML serialization directly from the structure of the underlying query. The XML serialization code uses Microsoft Dynamics AX concepts such as extended data types (EDTs) to further restrict valid XML schemas and improve XML schema validation. Moreover, when generating XML schemas, *Axd*<Document> classes take the data access layer features that have been introduced in Microsoft Dynamics AX 2012 into consideration. For example, the generated XML schema definitions reflect date-effective table fields, expanded dimension fields, and the inheritance structure of the tables used in the underlying *Axd* query, if applicable; surrogate foreign key fields are replaced with alternate keys, if configured.

Axd<Document> classes always access tables through the *Ax*<Table> classes. During serialization, *Axd*<Document> classes rely on *AxCommon* or custom *Ax*<Table> classes to persist data to tables and to read data from tables.

Figure 12-2 illustrates the mapping between a Microsoft Dynamics AX query used for the *Axd*<Document> class *AxdSalesOrder* and the generated XML schema definition.

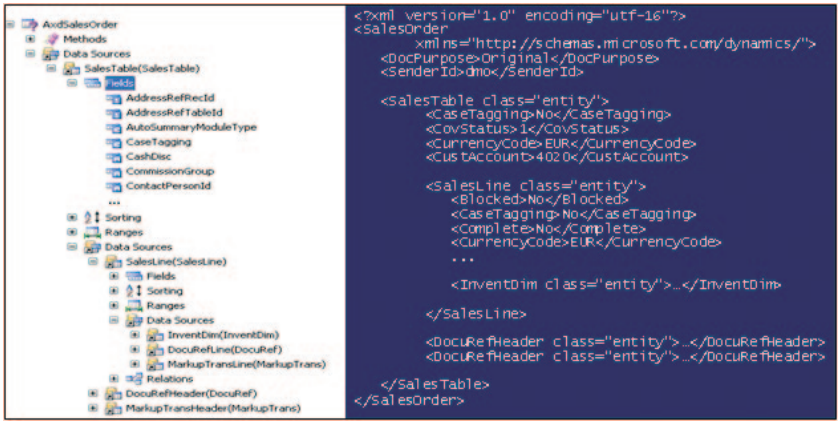


FIGURE 12-2 Correlation between the AOT query and the XML document structure.

Axd<Document> classes also provide an API for orchestrating cross-table field validation and defaulting. Validation and defaulting logic that is relevant only for a specific *Axd*<Document> class, but not for all *Axd*<Document> classes that use the same table, can also be implemented in *Axd*<Document> classes.

Axd<Document> instances can be uniquely identified through *AifEntityKeys*, which consist of a table name (name of the root table for the *Axd* query), the field names for a unique index of that table, and the values of the respective fields for the retrieved record. In addition, *AifEntityKeys* holds the record ID of the retrieved records.

Ax<Table> classes

Ax<Table> classes (such as *AxSalesTable* and *AxSalesLine*) derive from the X++ class *AxInternalBase*. Unlike in earlier versions of Microsoft Dynamics AX, an *Ax*<Table> class is not needed for each table that is used in a document service; instead, the *Ax*<Table> class *AxCommon* has been introduced in Microsoft Dynamics AX 2012, which *Axd*<document> classes use by default to access tables.

Note Document services that are included with Microsoft Dynamics AX 2012 might still rely on custom *Ax*<Table> classes for tables used in the underlying query, especially, if those services were created in earlier versions of Microsoft Dynamics AX, before the introduction of the *AxCommon* class.

However, there are scenarios in which custom *Ax*<Table> classes are required; for example, when *parm* methods for fields on the underlying table are needed to do the following:

- Support calculated fields for a table in the *Ax*<Table> class.
- Support a custom value mapping, which is different from the default implementation in *AxCommon*.

Note *Ax*<Table> classes are often referred to as *AxBC* classes in both code and documentation.

Although optional in Microsoft Dynamics AX 2012, *Ax*<Table> classes can be generated as part of the document service with the AIF Document Service Wizard.

Create document services

You generate document services based on *Axd* queries by using the AIF Document Service Wizard. This section discusses a few selected aspects of generating and maintaining document services.

Create *Axd* queries Although general guidelines for working with Microsoft Dynamics AX queries apply to *Axd* queries, some additional constraints and guidelines apply:

- Name Microsoft Dynamics AX queries that are used for document services with the prefix *Axd* followed by the document name. For example, the document service query for the document *SalesOrder* should be *AxdSalesOrder*. This is a best practice.
- Only one root table for each query is allowed. You can associate the unique entity key that is used to identify document instances with this root table. For example, the entity key *SalesId* is defined on the *AxdSalesOrder* root table *SalesTable*.
- If your query's data sources are joined by an inner join, you should use fetch mode 1:1; if they are joined by an outer join, you should use fetch mode 1:*n*. If you don't use these settings, your query and the service operations that use this query can yield unexpected results.

- If you want to use a Microsoft Dynamics AX document service to write data back to the database—that is, if you need to support the service operation *update*—set the AOT property *Update* to *Yes* for all data sources that the query uses to generate the service.

Note For security reasons, checks in X++ code by default prevent system tables from being used in queries that are used for document services.

Generate a document service To generate a document service from an existing *Axd* query, you can use the AIF Document Service Wizard. To start the wizard, on the Tools menu, point to Application Integration Framework > Create Document Service. This section provides a high-level description of the AIF Document Service Wizard and some important notes about how to use it.

In the wizard, you can select the service operations you want to generate for your service: *create*, *read*, *update*, *delete*, *find*, *findKeys*, *getKeys*, and *getChangedKeys*. If you select *Generate AxBC classes* when running the wizard, the wizard generates new *Ax*<Table> classes with *parm* methods for the fields of the tables used in the query.

The AIF Document Service Wizard uses the document name—which you enter on the first screen—to derive names for the generated artifacts. You can change the document name (and thus the derived names for the artifacts) in the wizard before the artifacts are generated. Names of AOT objects are limited to 40 characters. If you choose a document name that produces names that are too long for one or more artifacts, you may get error messages.

Once the wizard finishes, it displays a report of all generated artifacts and any errors encountered. You need to fix all errors before you start customizing the code that the wizard generates.

Tip The wizard creates a new project for each generated service. It then adds the generated artifacts automatically to the created project.

You can use the Update Document Service dialog box to update existing document services; for example, to add a service operation that you had not selected initially.

Note Although you can create and update document services manually, it is not recommended. Instead, always use the AIF Document Service Wizard to generate new document services from AOT queries and the Update Document Service dialog box quickly to update existing document services.

Microsoft Dynamics AX 2012 includes over 100 ready-to-use document services. These include services such as *SalesOrderService* and *CustomerService*. You can find a list of these services in the AOT *Services* node, or in the topic "Standard Document Services" in the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/aa859008.aspx*).

For a more comprehensive discussion of the AIF Document Service Wizard and generating *Axd*<Document> and *Ax*<Table> classes, see the "AIF Document Services" section of the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/bb496530.aspx*).

Customize document services

In many cases, you might need to customize the document services that you have generated from queries or that are included with Microsoft Dynamics AX 2012 to better fit your business needs. This section touches on some of the most common scenarios for customizing document services, including customizing the tables or queries, service operations, validation, defaulting, queries, and security.

Customize tables When you customize a table that is used by a document service (for example, by adding a column), you need to update the service implementation—that is, the *Axd*<Document> and *Ax*<Table> classes and the data objects—to reflect these changes.

Tip Always enable best practice checks with the Best Practices tool to detect potential discrepancies between the table structure and the service implementation. If the best practice checks on any of your customized tables fail, you can use the Update Document Service dialog box to update the *Axd*<Document> class, *Ax*<Table> classes, and data objects to reflect the changes.

Caution Because document services are based on Microsoft Dynamics AX queries, changes in the structure of a query that is used in a document service (for example, by adding a column to a table used in the query) inadvertently changes the data contract for that document service. Changes in external interfaces such as service interfaces can potentially break integrations that were built using the original data contract. Always consider the impact of changing queries or tables that are used in document services and apply common best practices for non-breaking service interface changes, such as not removing service operations, or data contract fields, and only adding optional fields.

Tip If you use a static field list for the query from which an *Axd* document service is generated, you can prevent the data contract for the *Axd* document service from implicitly changing when a field is added to a table.

Add custom service operations You can change the behavior of any service operation by modifying its X++ implementation. In addition, you can add custom service operations to any document service by following the same steps used for adding service operations to custom services.

Customize validation logic Validation logic is crucial for enforcing data hygiene. Ideally, invalid data never is persisted in the Microsoft Dynamics AX data store.

Tip To achieve this goal, always verify the validation logic of each service operation that you generate or customize to make sure that it meets your requirements.

Well-designed validation logic has the following characteristics:

- **Reusable** Ideally, the same (generic) validation logic can be used from the Microsoft Dynamics AX client and from Microsoft Dynamics AX services. Keep in mind that non-generic validation code, code that applies only to the Microsoft Dynamics AX client or only to Microsoft Dynamics AX services, is also possible.
- **Well-performing** Validation code runs whenever the respective Microsoft Dynamics AX entity is modified. As a consequence, one of your key goals for writing validation logic must be adequate performance.
- **Sufficient** Validation logic must guarantee a sufficient level of data hygiene. You might have to trade sufficiency for performance in a way that satisfies your application's requirements.

Validation code consists mainly of the following elements:

- Code that orchestrates cross-table validation by invoking validation code that is implemented on the respective tables. This code is implemented in the respective *Axd*<Document> class methods *prepareForSave*, *prepareForUpdate*, and *prepareForDelete*. These *prepareForXxx* methods are called once for each *Ax*<Table> class that the *Axd*<Document> class uses.
- Code that enforces table-level validation logic is implemented by the table methods *validateField* and *validateWrite* for maximum code reusability. These methods call specific validation methods, such as *checkCreditLimit* on *SalesTable*.
- Code that performs document-level validation, which is implemented by the *Axd*<Document> class method *validateDocument*. This method is called immediately before changes are persisted to tables, and after the *prepareForXxx* methods have been called for each *Ax*<Table> class.
- Code that performs validation after data has been persisted to the table, which is implemented by the *Axd*<Document> class method *updateNow*.

The following code, the *prepareForSave* method for *AxdSalesOrder*, is an example of cross-table validation. It calls validation methods for the *Ax*<Table> classes *AxSalesTable* and *AxSalesLine* (in addition to other *Ax*<Table> classes, which have been removed from this example):

```
public boolean prepareForSave(AxdStack _axdStack, str _dataSourceName)
{

    // ...

    switch (classidget(_axdStack.top()))
    {
        case classnum(AxSalesTable) :
            axSalesTable = _axdStack.top();
            this.checkSalesTable(axSalesTable);
```

```
            this.prepareSalesTable(axSalesTable);
            return true;

        case classnum(AxSalesLine) :
            axSalesLine = _axdStack.top();
            this.checkSalesLine(axSalesLine);
            this.prepareSalesLine(axSalesLine);
            return true;

        // ...
    }

    return false;
}
```

Customize defaulting logic You can customize the defaulting logic for table fields that is executed as part of creating or updating table rows. Defaulting logic helps increase the usability of both interactive client applications and Microsoft Dynamics AX service interfaces. It derives initial values for table fields from other data—such as values of other table fields—and thus, it doesn't require explicit value assignments for the defaulted table fields. It also helps reduce the amount of data required to manipulate more complex entities, such as sales orders, while lowering the probability of erroneous data entry.

Well-designed defaulting logic has the following characteristics:

- **Reusable** You should implement defaulting logic so that it is reusable—that is, so the same logic can be used regardless of which Microsoft Dynamics AX client (for example, a user interface or a service client) creates or updates the entity. In certain scenarios, the defaulting of table fields might require different logic, depending on whether the Microsoft Dynamics AX client is interactive (a user interface) or non-interactive (a request from a service client).
- **Well-performing** Because the defaulting logic for a table field is invoked every time the field is set, its execution time directly affects the processing time for manipulating the entity, such as a sales order. In particular, try to avoid redundant defaulting steps—that is, setting a field value that is overwritten again as part of the same defaulting logic.
- **Sufficient** To reduce the number of required fields for manipulating entities, as many fields as possible should be defaulted, while still meeting the performance goals.

Microsoft Dynamics AX 2012 still supports the approach to implementing defaulting logic that was supported in previous versions of Microsoft Dynamics AX. However, in Microsoft Dynamics AX 2012, mechanisms for tracking field states (such as *not set* and *defaulted*) have been added to tables, which means that you can implement defaulting logic directly in table classes. This allows for defaulting logic to be used not only by *Axd*<document> classes, but also from forms, and so on. Note that because now you can implement defaulting logic directly in the table class, an *Ax*<Table> class is not necessary for implementing standard defaulting code.

For more details about implementing and customizing defaulting logic in Microsoft Dynamics AX 2012 and information about how to customize document services in general, see the "AIF Document Services" section of the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/bb496530.aspx*).

Security considerations

Service operations are entry points through which external applications can submit requests on behalf of users. As mentioned earlier, all X++ methods that are intended to be published as service operations must be annotated with the X++ attribute *SysEntryPointAttribute*, indicating whether the method is to be invoked in the context of the calling user. If so, authorization checks must be performed for tables accessed within the method. In addition, all concepts related to role-based security also apply to services and service operations.

System services are generally accessible and executed in the calling user's context.

Because as the developer, you are in charge of the implementation of custom services, you must add the *SysEntryPointAttribute* manually to all service operations and create permissions when necessary.

When you generate document services by using the AIF Document Service Wizard, all generated service operations are automatically annotated with *SysEntryPointAttribute*. Moreover, the wizard attempts to infer all security permissions for the generated service automatically.

Tip When using the AIF Document Service Wizard, always verify that the generated artifacts meet your requirements and adjust them if they don't.

Publish Microsoft Dynamics AX services

After you create and customize your service, you need to publish it for external applications to be able to consume it. Developing a service and publishing a service are two separate and largely independent processes.

With the AIF, you can publish Microsoft Dynamics AX services through various transport technologies. In addition, the AIF provides a variety of configuration options that administrators can use to customize how service interfaces are published. This chapter limits the discussion of the AIF to publishing services through *basic integration ports*. For more information, see the services administration documentation for Microsoft Dynamics AX 2012 on TechNet (*http://technet.microsoft.com/en-us/library/hh209600.aspx*). You can also find guidance on how to develop, set up, and use concepts such as data policies, transformations, pipeline components, and value mappings.

For development and debugging purposes, you can easily publish registered custom services and document services through basic integration ports right from the AOT. You can also use service groups to ensure that services are deployed and activated automatically when the AOS is started by using the *AutoDeploy* property of the respective service group. This is useful when you need to be able to consume a service without administrator intervention; for example, to enable the service manually after deploying Microsoft Dynamics AX 2012.

To publish a service through a basic integration port, first, you need to add it to a *service group* in the AOT. Then you can deploy the service group with a default configuration using *NetTcpBinding* in WCF, right from the AOT. For more information, see the topics "Services, service operations, and service groups" (*http://technet.microsoft.com/en-us/library/gg731906.aspx*) and "Using basic integration ports" (*http://technet.microsoft.com/en-us/library/hh496420.aspx*) on TechNet.

Microsoft Dynamics AX services that are published through basic integration ports can only be hosted directly on the AOS. There are limited configuration options available for services published through basic integration ports. From the Inbound ports form (System Administration > Setup > Services And Application Integration Framework > Inbound Ports), you can activate and deactivate basic integration ports, you can use *SvcConfigUtil* to modify WCF configuration parameters, and you can enable logging for the respective ports.

Note If you need to publish a service through a WCF binding other than *NetTcpBinding*, if you need to send unsolicited messages (outbound messages), or if you need more control over message processing and, for example, use XSLT transformations, you must create an enhanced integration port. You can create *enhanced integration ports* from the Inbound Ports form or the Outbound Ports form, respectively.

Discussions in this chapter generally assume that services have been published through basic integration ports unless noted otherwise. For details about how to publish services through bindings other than *NetTcpBinding* (for example, Message Queuing or file system adapters) using enhanced integration ports, how to create ports for outbound messages, and for additional configuration options, see the services and AIF documentation for Microsoft Dynamics AX 2012 on TechNet (*http://technet.microsoft.com/en-us/library/gg731810.aspx*).

Consume Microsoft Dynamics AX services

After you publish your Microsoft Dynamics AX services, external client applications can consume them and invoke the exposed business logic. For example, once the *SalesOrderService* is exposed, client applications can consume it to create or read Microsoft Dynamics AX sales orders.

This section highlights a few aspects of consuming Microsoft Dynamics AX services from client applications. As mentioned earlier, this chapter assumes that services are published through basic integration ports on the AOS. Services that are published through basic integration ports are accessible through Net.tcp.

For a more complete description of ways of publishing Microsoft Dynamics AX services, including the use of asynchronous adapters and related technologies, see the services and AIF documentation for Microsoft Dynamics AX 2012 on TechNet (*http://technet.microsoft.com/en-us/library/gg731810.aspx*).

Sample WCF client for *CustCustomerService*

If you want to consume a Microsoft Dynamics AX service that has been published through a basic integration port, you need to generate proxy classes from the WSDL of the service you want to consume. Typically, you do this either from within your development environment (Microsoft Visual Studio) or by using a command-line tool such as *SvcUtil*.

After you generate the proxy classes from the WSDL and add them to a project in your development environment, you need to write code to do the following:

- Instantiate and initialize parameters.
- Optionally instantiate and initialize a call context.
- Instantiate a service proxy.
- Consume the service operation.
- Evaluate the response.
- Handle errors and exceptions.

This section contains an example that illustrates what the code for consuming the service operation *find()* on the document service *CustCustomerService* (included with Microsoft Dynamics AX 2012) might look like.

For the following examples, assume that the document service *CustCustomerService* has been published through the service group *MyServiceGroup* and a Visual Studio project has been created. Also, in Visual Studio, the service reference *MyServiceGroup* was added by using the WSDL for the basic integration port *MyServiceGroup*. For details about where Microsoft Dynamics AX publishes WSDL files, see the topic "Locating the WSDL for Services" in the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/gg843514.aspx*).

The following code snippets show C# code for the steps to consume the service operation *find* of the Microsoft Dynamics AX document service *CustCustomerService*.

First, you need to instantiate and initialize the parameters needed for the call. The service operation *find* accepts two input parameters: an optional call context and a query criterion that specifies which customer records should be returned. The following example retrieves all customer records in the company CEU with an account number greater than or equal to 4,000:

```
// instantiate and initialize parameters

// parameter: call context
MyServiceGroup.CallContext cc = new MyServiceGroup.CallContext();
cc.Company = "CEU";

// parameter: query criteria
MyServiceGroup.QueryCriteria qc = new MyServiceGroup.QueryCriteria();
MyServiceGroup.CriteriaElement[] qe = { new MyServiceGroup.CriteriaElement() };
qe[0].DataSourceName = "CustTable";
```

```
qe[0].FieldName = "AccountNum";
qe[0].Operator = MyServiceGroup.Operator.GreaterOrEqual;
qe[0].Value1 = "4000";
qc.CriteriaElement = qe;
```

Tip You can use a *CallContext* object to execute a request in a different context than the default context, which is used if a null or empty *CallContext* object is used for a request. In the *CallContext* object, you can specify the company, language, and more.

Next, you need to instantiate a service proxy and consume the service operation *find*, which executes a query and returns matching entities:

```
// instantiate a service proxy
MyServiceGroup.CustomerServiceClient customerService =
    new MyServiceGroup.CustomerServiceClient();

// consume the service operation find()
MyServiceGroup.AxdCustomer customer = customerService.find(cc, qc);
```

Finally, you need to evaluate the response from the server, which can be either query results or exception and error messages:

```
// error handling (additionally, exceptions need to be handled properly)
if (null == customer)
{
    // error handling...
}

// evaluate response
MyServiceGroup.AxdEntity_CustTable[] custTables = customer.CustTable;
if (null == custTables || 0 == custTables.Length)
{
    // handle empty response...
}
foreach (MyServiceGroup.AxdEntity_CustTable custTable in custTables)
{
    custTable...
}
```

Note Exception handling and other common best practices for developing web service clients are omitted from the simplified code examples.

Here are some tips for working with document services:

- Many document services support both service operations *find* (which returns all *Axd* documents in the result set) and *findKeys* (which returns only the entity keys for *Axd* documents in the result set). If you expect the response message for invoking *find* to be very large, you might want to use *findKeys* to retrieve the entity keys. You can then, for example, implement paging to retrieve the matching *Axd* documents in sizeable chunks.

- When developing new services, it is usually useful to turn on logging on the server side. To do that, open the Inbound Ports form, deactivate the integration port that publishes the service group containing your service, enable logging in the Troubleshooting section of the Inbound Ports form, and then reactivate your integration port.
- If a service operation returns large response messages, you may need to tweak the default settings in your WCF configuration files for both the service and the client. By default, both service and client WCF configurations allow messages of sizes up to 65,536 bytes. The maximum message and buffer sizes are defined through the parameters *maxReceivedMessageSize* and *maxBufferSize* in the binding section of standard WCF configuration files. Before changing these parameters, refer to .NET Framework developer documentation to understand implications and valid values for these parameters. The .NET Framework Developer Center is located at *http://msdn.microsoft.com/en-us/netframework/aa496123*.

Other service operations for custom or document services can be consumed in similar ways. For more information and code examples, see the Microsoft Dynamics AX 2012 SDK at *http://msdn.microsoft.com/en-us/library/aa496079.aspx*.

Consume system services

Unlike custom services and document services, system services are automatically published (on the AOS by using the *NetTcpBinding*) and are ready for consumption by client applications when AOS starts.

Like all Microsoft Dynamics AX services, system services publish metadata in the form of WSDL files, which you can use for proxy generation (see the previous examples). However, while the user session info service is published explicitly through an integration port (*UserSessionService*), similar to custom and document services, an integration port does not exist for the query service or the metadata service.

To provide an example of how to work with the metadata service and the query service, the following code example shows how to do the following:

- Retrieve query metadata—the definition of a query named *MyQuery*—from Microsoft Dynamics AX by using the metadata service.
- Convert the query metadata from the data contract used by the metadata service to the data contract used by the query service. This conversion is necessary although both data contracts are structurally identical (see the method *ConvertContract* in the following code example).
- Add a range to the metadata object; in this case, include all rows with a value greater than 1996 for the *Year* column.
- Execute the converted query definition by using the query service.

In .NET code, these steps could be implemented in a similar way to the code sample that follows. Assume that you've created a Visual Studio project and added the references *MetadataService* and *QueryService* by using the WSDLs for the metadata service and the query service, respectively.

For details about where Microsoft Dynamics AX publishes WSDL files, see the topic "Locating the WSDL for Services" in the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/gg843514.aspx*).

```
// instantiate proxies
var metadataClient = new MetadataServiceReference.AxMetadataServiceClient();
var queryClient = new QueryServiceReference.QueryServiceClient();

// retrieve query metadata
MetadataService.QueryMetadata[] query =
  metadataClient.GetQueryMetadataByName(new string[] { "MyQuery" });

// convert query metadata
QueryService.QueryMetadata convertedQuery = ConvertContract
  <MetadataService.QueryMetadata, QueryService.QueryMetadata>(query);

// add a range to the query metadata object
QueryDataRangeMetadata range = new QueryDataRangeMetadata()
{
  Enabled = true,
  FieldName = "Year",
  Value = ">1996"
};
convertedQuery.DataSources[0].Ranges = new QueryRangeMetadata[] { range };

// initialize paging (return 3 records or less)
QueryService.Paging paging = new QueryService.ValueBasedPaging();
((QueryService.ValueBasedPaging)paging).RecordLimit = 3;

// instantiate a service proxy
QueryService.QueryServiceClient queryService =
  new QueryService.QueryServiceClient();

// execute the converted query with the range, receive results into .NET dataset
System.Data.DataSet ds =
  queryClient.ExecuteQuery(convertedQuery, ref paging);
```

Note that although the *QueryMetadata* definition is identical in both the query service and the metadata service, the proxy generator generates an identical class in two different namespaces, one for each service. A *ConvertContract* method that implements the conversion of two contracts of the same structure by using generics could look similar to the following code:

```
static TTargetContract ConvertContract<TSourceContract, TTargetContract>
    (TSourceContract sourceContract)
        where TSourceContract : class
        where TTargetContract : class
{
  TTargetContract targetContract = default(TTargetContract);
  var sourceSerializer = new DataContractSerializer(typeof(TSourceContract));
  var targetSerializer = new DataContractSerializer(typeof(TTargetContract));
  using (var stream = new MemoryStream())
```

```
    {
      sourceSerializer.WriteObject(stream, sourceContract);
      stream.Position = 0;
      targetContract = (TTargetContract)targetSerializer.ReadObject(stream);
    }
    return targetContract;
  }
```

As mentioned earlier, the *CallContext* is used to override the default context (such as company and language) in which a request is executed. A *CallContext* is optional for all service requests; if it is not present in a request, the request is executed using default values for the *CallContext* properties.

In Microsoft Dynamics AX 2012, the WSDL files for the query service and the metadata service do not contain the XML schema definitions for *CallContext*. Consequently, proxies generated from the WSDL files for those services do not include proxy classes for *CallContext*; however, *CallContext* can still be used for the query service and the metadata service the same way as it is with other services. To use *CallContext* in requests sent to the metadata service or the query service, you need to add a service reference to an integration port (such as *UserSessionService*), which generates the proxy classes necessary for *CallContext*. You can then instantiate and initialize a *CallContext* object and add it to your request, as shown in the following code:

```
// get OperationContextScope (see WCF documentation)
using (System.ServiceModel.OperationContextScope ocs =
    new System.ServiceModel.OperationContextScope((queryService.InnerChannel))) {

  // instantiate and initialize CallContext (using class from other service)
  CustomerService.CallContext callContext = new CustomerService.CallContext();
  callContext.Company = "CEU";

  // explicitly add header "CallContext" to set of outgoing headers
  System.ServiceModel.Channels.MessageHeaders messageHeadersElement =
  System.ServiceModel.OperationContext.Current.OutgoingMessageHeaders;
  messageHeadersElement.Add(
    System.ServiceModel.Channels.MessageHeader.CreateHeader(
      "CallContext",
      "http://schemas.microsoft.com/dynamics/2010/01/datacontracts",
      callContext));

  // initialize paging (return 3 records or less)
  QueryService.Paging paging = new QueryService.ValueBasedPaging();
  ((QueryService.ValueBasedPaging)paging).RecordLimit = 3;

  // instantiate a service proxy
  QueryService.QueryServiceClient queryService =
      new QueryService.QueryServiceClient();

  // consume query service using CallContext
  System.Data.DataSet ds =
      queryService.ExecuteStaticQuery("MyQuery", ref paging);
}
```

Note The query service returns query results in chunks that are defined through a required paging parameter. The paging algorithms assume that queries use relations with *FetchMode* set to 1:1 (AOT property). The query service produces an error message for queries that use relations with *FetchMode* set to 1:*n*.

Please refer to the product documentation for further details on the *CallContext* or capabilities of system services.

Update business documents

In many scenarios, you need to update data in already-existing *Axd* documents, such as to add a sales line to a sales order or to update a customer address. Through the service operation *update*, document services support different semantics for document-centric updates: *full updates* and *partial updates*.

For the following examples, assume that the standard document service *SalesSalesOrderService* has been added to a service group named *MyServiceGroup* and published through a basic integration port named *MyServiceGroup*.

Apply a full update

Full updates are the default behavior for document services. To use this mode, add code to your client application to do the following:

- Read the document.
- Apply changes to the document.
- Send the updated document back to the server.
- Handle errors, if any.

The following C# code provides a conceptual example of how to apply a full update to an existing sales order:

```
// instantiate and initialize callContext, entityKeys, serviceOrderService
MyServiceGroup.EntityKey[] entityKeys = ...
MyServiceGroup.CallContext callContext = ...
MyServiceGroup.SalesOrderServiceClient salesOrderService = ...
...

// read sales order(s) (including document hash(es)) using entityKeys
MyServiceGroup.AxdSalesOrder salesOrder =
    salesOrderService.read(callContext, entityKeys);

// handle errors, exceptions; process sales order, update data
...
```

```
// persist updates on the server (exception handling not shown)
salesOrderService.update(callContext, entityKeys, salesOrder);
```

Apply a partial update

In many scenarios, full updates are inefficient. Imagine a large sales order with many sales lines—having more than 1,000 is not uncommon. If you use a full update, you would have to retrieve the entire sales order with all sales lines, apply your changes to the one sales line you want to update, and then send back the entire sales order—including all unchanged sales lines. This operation can be costly when you consider the validation and defaulting logic invoked on the server for each sales line.

Instead of performing a full update, you can apply a partial update. Partial updates use the same service operation as full updates do: *update*. However, with partial updates, you can send partial documents that contain only the changed (added, modified, or deleted) data. For child elements, documents sent in partial update requests contain processing instructions specifying how to handle each (child) record included in the partial document to avoid ambiguity. Consequently, the process for updating documents by using partial updates contains one additional step:

- Read the document.
- Apply changes to the document. To take advantage of partial updates, ensure that you only send the fields back to the server that are either mandatory or that have changed.
- Explicitly request the partial update mode and add processing instructions.
- Send the updated document with the update request.
- Handle errors, if any.

The following code provides a conceptual example of how to apply a partial update to a sales order:

```
// instantiate and initialize callContext, entityKeys, serviceOrderService
MyServiceGroup.EntityKey[] entityKeys = ...
MyServiceGroup.CallContext callContext = ...
MyServiceGroup.SalesOrderServiceClient salesOrderService = ...
...

// read sales order(s) (including document hash(es)) using entityKeys
MyServiceGroup.AxdSalesOrder salesOrder =
    salesOrderService.read(callContext, entityKeys);

// handle errors, exceptions; process sales order, update data
...

// example: update the first sales order and mark it for partial update
AxdEntity_SalesTable[] salesTables = salesOrder.SalesTable;
salesOrder.SalesTable = new AxdEntity_SalesTable[] { salesTables[0] };
// document-level directive, requesting a partial update
salesOrder.SalesTable[0].action = AxdEnum_AxdEntityAction.update;
```

```
// table-level directive, requesting to delete the first sales line
AxdEntity_SalesLine[] salesLines = salesOrder.SalesTable[0].salesLine;
salesOrder.SalesTable[0].SalesLine = new AxdEntity_SalesLine[] { salesLines[0] };
salesOrder.SalesTable[0].SalesLine[0].action = AxdEnum_AxdEntityAction.delete;

// remove child data sources w/o updates (DocuRefHeader, etc.) from salesTable
...

// persist updates on the server (exception handling not shown)
salesOrderService.update(callContext, entityKeys, salesOrder);
```

Note In XML request messages, these processing instructions are reflected through occurrences of the XML attribute *action*. This is true for both XML messages sent to asynchronous adapters and for Simple Object Access Protocol (SOAP) messages sent to synchronous WCF services. For more details, see the "AIF Document Services" section of the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/bb496530.aspx*)

Optimistic concurrency control

The services framework relies on optimistic concurrency control (OCC) to resolve conflicts when multiple concurrent update requests occur. To be able to detect whether a document has changed since it was last read, and to avoid inadvertently overwriting such changes, the service framework uses document hashes to identify versions of a business document.

Document hashes are computed for a specific document instance from its contents; they are derived not only from the root-level data source (such as the sales header) but also from all of the joined data sources (such as a sales line). In other words, if a field in any table that is included in the business document changes, the document hash changes too.

To obtain the document hash for a business document, your code must first read the document. It can then use the document hash that was returned inside the document in a subsequent update request.

Tip Caching a document for a long time on a service client without refreshing it increases the probability of update requests being rejected because of colliding updates from other client applications.

Invoke custom services asynchronously

Because publishing a service is separate from developing the service, both custom services and document services can be published through the supported transport mechanisms; more specifically, a custom or document service's operations can be published synchronously (for example, by using the Net.tcp or HTTP protocol) through basic integration ports, as shown in the previous examples, or they can be published asynchronously (for example,. by using the file system adapter or Message

Queuing) through enhanced integration ports. Administrators can select various options to configure how service operations are bundled and published at run time and to configure logging, among other things. For more information about publishing services through enhanced integration ports, see the services and AIF documentation for Microsoft Dynamics AX 2012 on TechNet (*http://technet.microsoft.com/en-us/library/gg731810.aspx*).

When consuming Microsoft Dynamics AX services synchronously, typically generated service proxies take care of producing and consuming the XML that is exchanged between the client application and Microsoft Dynamics AX. However, when consuming Microsoft Dynamics AX services through asynchronous transports, you need to make sure that the request messages comply with the XML schema definitions for the AIF message envelope and the business document as expected by the Microsoft Dynamics AX service framework. For more information about how to get the XML schema definitions (XSDs) for message envelopes, see the "AIF Messages" section in the Microsoft Dynamics AX 2012 SDK (*http://msdn.microsoft.com/en-us/library/aa627117.aspx*)

The following code example shows a sample XML message that can be sent asynchronously from a client application to Microsoft Dynamics AX to consume the service operation *MyService.HelloWorld(MyParam in)* of a custom service was discussed in a previous example (see the "Custom services" section, earlier in this chapter). It illustrates how the service name, the service operation name, and the structure of the input parameters map to the corresponding elements of the XML request message. It also shows how you can specify the context in which the request is executed: through the *Header* element, which recognizes the same properties the *CallContext* knows in the case of synchronous service interfaces:

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope
    xmlns="http://schemas.microsoft.com/dynamics/2011/01/documents/Message">
  <Header>
    <!-- Service operation: "MyService.HelloWorld(MyParam)" -->
    <Company>CEU</Company>
    <Action>http://tempuri.org/MyService/HelloWorld</Action>
  </Header>
  <Body>
    <MessageParts
        xmlns="http://schemas.microsoft.com/dynamics/2011/01/documents/Message">

      <!-- Complex input parameter: "MyParam in" -->
      <in xmlns="http://tempuri.org"
          xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:b="http://schemas.datacontract.org/2004/07/Dynamics.Ax.Application">

        <!–Property of complex input parameter: "in.b" -->
        <b:intParm>0</b:intParm>
      </in>
    </MessageParts>
  </Body>
</Envelope>
```

Note To run this example, you need to create an enhanced integration port that is configured to receive files asynchronously. That integration port must publish the service operation *MyService.HelloWorld*.

So far, this chapter has discussed how Microsoft Dynamics AX functionality can be published through services for consumption by external client applications and how external client applications can consume these services. But what if you want to send unsolicited data out of Microsoft Dynamics AX? The following two sections discuss how to use the Microsoft Dynamics AX send framework for sending unsolicited data asynchronously.

The Microsoft Dynamics AX send framework

AIF provides APIs and infrastructure for using Microsoft Dynamics AX services to send unsolicited one-way messages. The Microsoft Dynamics AX client has features like the Send Electronically button on several forms that allow users to transmit business documents (such as invoices) as unsolicited one-way messages through outbound integration ports. For information about how to configure outbound integration ports, see the services and AIF documentation for Microsoft Dynamics AX 2012 on TechNet (*http://technet.microsoft.com/en-us/library/gg731810.aspx*).

Microsoft Dynamics AX doesn't rely on external document schema definitions to be provided by the remote receiving application; it uses its own format instead—the same *Axd*<Document> class-based XSDs that are also used as data contracts for published Microsoft Dynamics AX services.

Implementing unsolicited one-way messages requires the following two steps:

- Implement a trigger for transmission (design time).
- Configure an enhanced outbound integration port for sending documents (administration time).

Implementing a trigger for transmission

You can implement a trigger for transmission by using either the AIF Send API or the AxdSend API.

AIF Send API

The Send API features a set of methods that can be used to send unsolicited one-way messages from Microsoft Dynamics AX by means of integration ports, through which the consumers can pick up the messages. This API sends a single message: the body of the message contains the XML that is generated by invoking the *read* service operation of the AIF document service referenced by the *serviceClassId* (it must reference a class that derives from *AifDocumentService*) with the parameter *entityKey*.

To see a working example of how you can use this API, look at the code behind the method *clicked* for the button SendXmlOriginal on the form CustInvoiceJournal. The API methods are defined on the class *AifSendService* and include the method *submitDefault*:

```
public static void submitDefault(
    AifServiceClassId serviceClassId,
    AifEntityKey entityKey,
    AifConstraintList constraintList,
    AifSendMode sendMode,
    AifPropertyBag propertyBag = connull(),
    AifProcessingMode processingMode = AifProcessingMode::Sequential,
    AifConversationId conversationId = #NoConversationId
)
```

By using the two optional parameters in the preceding signature, *processingMode* and *conversationId*, you can take advantage of the parallel message processing feature for asynchronous adapters:

- ***processingMode*** Specifies whether messages can be moved from the AIF outbound processing queue to the AIF gateway queue in parallel (*AifProcessingMode::Parallel*) or whether first-in-first-out (FIFO) order must be enforced for all messages (*AifProcessingMode::Sequential*).
- ***conversationId*** If this is specified, AIF moves the message from the AIF outbound processing queue to the AIF gateway queue in FIFO order, relative to all other messages with the same *conversationId*. The order relative to other messages with different *conversationIds* isn't guaranteed.

AxdSend API

The AxdSend API provides functionality to send unsolicited one-way messages. The user selects the outbound integration port through which the documents are sent at run time. If more than one document needs to be sent, the user also selects the exact set of entities at run time. This feature has been implemented for several Microsoft Dynamics AX document services, such as *AxdPricelist* and *AxdBillsOfMaterials*.

The *AxdSend* framework provides default dialog boxes for selecting integration ports and entity ranges and allows the generation of XML documents with multiple records. You can use the framework to provide specific dialog boxes for documents that require more user input than the default dialog box provides.

The default dialog box includes an integration port drop-down list and, optionally, a Select button to open the standard query form. The query is retrieved from the *Axd*<Document> class that the caller specifies. Many integration ports can be configured in AIF, but only a few are allowed to receive the current document. The lookup shows only the integration ports that are valid for the document, complying with the constraint set up for the *read* service operation for the current document.

The framework requires minimal coding to support a new document. If a document requires the user to just select an integration port and fill out a query range, most of the functionality is provided by the framework without requiring additional code.

An example dialog box for the *AxdSend* framework is shown in Figure 12-3.

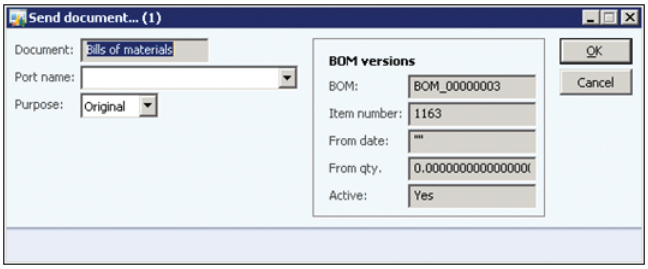


FIGURE 12-3 The Send Document Electronically dialog box for bills of materials.

If an *Axd*<document> requires a more specific dialog box, you inherit the *AxdSend* class and provide the necessary user interface interaction to the dialog box method. In the following code example, an extra field has been added to the dialog box. You just add one line of code to implement *parmShowDocPurpose* from the *AxdSend* class and to make this field appear on the dialog box:

```
static public void main(Args args)
{
   AxdSendBillsOfMaterials axdSendBillsOfMaterials;
   AifConstraintList       aifConstraintList;
   AifConstraint           aifConstraint;
   BOMVersion              bomVersionRecord;

   axdSendBillsOfMaterials = new  AxdSendBillsOfMaterials();
   aifConstraintList       = new AifConstraintList();
   aifConstraint           = new AifConstraint();

   aifConstraint.parmType(AifConstraintType::NoConstraint);
   aifConstraintList.addConstraint(aifConstraint);

   if (args && args.record().TableId == tablenum(BOMVersion))
   {
       bomVersionRecord = args.record();
       axdSendBillsOfMaterials.parmBOMVersion(bomVersionRecord);
   }

   // added line to make the field appear on the dialog box
   axdSendBillsOfMaterials.parmShowDocPurpose(true) ;

   axdSendBillsOfMaterials.sendMultipleDocuments(
      classnum(BomBillsofMaterials),
      classnum(BomBillsofMaterialsService),
      AifSendMode::Async,
      aifConstraintList);
}
```

Sorting isn't supported in the *AxdSend* framework, and the query structure is locked to ensure that the resulting query matches the query defined by the XML document framework. Because of this need for matching, the *AxdSend* class enforces these sorting and structure limitations. The query dialog box shows only the fields in the top-level tables because of the mechanics of queries with an

outer join predicate. The result set will likely be different from what a user would expect. For example, restrictions on inner data sources filter only these data sources, not the data sources that contain them. The restrictions are imposed on the user interface to match the restrictions on the query when using the document service's *find* operation.

Configure transmission mechanisms

For details about configuring enhanced outbound integration ports and other administrative features related to sending unsolicited messages asynchronously by using the Microsoft Dynamics AX send framework, see the services and AIF documentation for Microsoft Dynamics AX 2012 on TechNet (*http://technet.microsoft.com/en-us/library/gg731810.aspx*).

Consume external web services from Microsoft Dynamics AX

Web services are a popular and well-understood way of integrating applications that are deployed within an enterprise's perimeter, or intranet. Examples of such applications include enterprise resource planning (ERP) applications, CRM applications, productivity applications such as Office, and so on.

Integrating applications with third-party web services over the Internet has also become viable and in many cases is the preferred approach for quickly adding new functionality to complex applications. Web services can range from simple address validation or credit card checks to more complex tax calculations or treasury services.

Similar to sending unsolicited data asynchronously by using the Microsoft Dynamics AX send framework, you can customize Microsoft Dynamics AX to send requests to external web services—in other words, to consume external web services. Because consuming external web services implies a tight coupling with the respective web service (and usually involves a service proxy for the web service), and because Visual Studio provides a rich set of tools for building such integrations, you should create a Visual Studio project and build a .NET dynamic link library (DLL) that contains the code to consume the external web service. You can then add this library as a reference to Microsoft Dynamics AX and write X++ code that calls methods exposed by this .NET library.

Note The Microsoft Dynamics AX service framework does not provide any tools specific to writing code to consume external web services. The concept of service references as it existed in Microsoft Dynamics AX 2009 has been removed from Microsoft Dynamics AX 2012, and the related AOT node no longer exists.

Performance considerations

To meet performance requirements for a specific Microsoft Dynamics AX implementation scenario, planning for and sizing of the hardware infrastructure is critical. For guidance on how to size your deployment properly, see the *Microsoft Dynamics AX Implementation Planning Guide* (*http://www.microsoft.com/en-us/download/details.aspx?id=4007*).

By default, integration ports process all request messages in sequence. This is true for both incoming and outgoing request messages. To increase the number of request messages that can be processed, you can use the AIF parallel processing capabilities in combination with additional AOS instances. For more information about how to configure inbound ports for parallelism and how to use extensions to the AIF Send API, see the "Services and AIF Operations" section of the Microsoft Dynamics AX 2012 system administrator documentation on TechNet (*http://technet.microsoft.com/en-us/library/gg731830.aspx*).

Note that for synchronous WCF services, request processing is inherently parallel.

CHAPTER 13

Performance

In this chapter

Introduction 417
Client/server performance 417
Transaction performance 426
Performance configuration options 462
Coding patterns for performance 465
Performance monitoring tools 478

Introduction

Performance is often an afterthought for development teams. Often, performance is not considered until late in the development process or, more critically, after a customer reports severe performance problems in a production environment. After a feature is implemented, making more than minor performance improvements is often too difficult. But if you know how to use the performance optimization features in Microsoft Dynamics AX, you can create designs that allow for optimal performance within the boundaries of the Microsoft Dynamics AX development and run-time environments.

This chapter discusses some of the most important facets of optimizing performance, and it provides an overview of performance configuration options and performance monitoring tools. For the latest information about how to optimize performance in Microsoft Dynamics AX, check the Microsoft Dynamics AX Performance Team blog at *http://blogs.msdn.com/axperf*. The Performance Team updates this blog regularly with new information. Specific blog entries are referenced throughout this chapter to supplement the information provided here.

Client/server performance

Client/server communication is one of the key areas that you can optimize for Microsoft Dynamics AX. This section details the best practices, patterns, and programming techniques that yield optimal communication between the client and the server.

Reduce round-trips between the client and the server

The following three techniques can help reduce round-trips significantly in many scenarios:

- Use the *cacheAddMethod* method for all relevant display and edit methods on a form, along with declarative display method caching.
- Refactor *RunBase* classes to support marshaling of the dialog box between the client and the server.
- Use proper caching and indexing techniques.

The *cacheAddMethod* method

Display and edit methods are used on forms to display data that must be derived or calculated based on other information in the underlying table. These methods can be written on either the table or the form. By default, these methods are calculated one by one, and if there is a need to go to the server when one of these methods runs, as there usually is, each function goes to the server individually. The fields associated with these methods are recalculated every time a refresh is triggered on the form, which can occur when a user edits fields, uses menu items, or presses F5. Such a technique is expensive in both round trips and the number of calls that it places to the database from the Application Object Server (AOS).

Caching cannot be performed for display and edit methods that are declared on the data source for a form because the methods require access to the form metadata. If possible, you should move these methods to the table. For display and edit methods that are declared on a table, use the *FormDataSource.cacheAddMethod* method to enable caching. This method allows the form's engine to calculate all the necessary fields in one round-trip to the server and then cache the results. To use *cacheAddMethod*, in the *init* method of a data source that uses display or edit methods, call *cacheAddMethod* on that data source and pass in the method string for the display or edit method. For example, look at the *SalesLine* data source of the *SalesTable* form. In the *init* method, you will find the following code:

```
public void init()
{
    super();
    salesLine_ds.cacheAddMethod(tableMethodStr(SalesLine, invoicedInTotal), false);
    salesLine_ds.cacheAddMethod(tableMethodStr(SalesLine, deliveredInTotal), false);
    salesLine_ds.cacheAddMethod(tableMethodStr(SalesLine, itemName), false);
    salesLine_ds.cacheAddMethod(tableMethodStr(SalesLine, timeZoneSite), true);
}
```

If you were to remove this code with comments, each display method would be computed for every operation on the form data source, increasing the number of round-trips to the AOS and the number of calls to the database server. For more information about *cacheAddMethod*, see *http://msdn.microsoft.com/en-us/library/formdatasource.cacheaddmethod.aspx*.

Note Do not register display or edit methods that are not used on the form. Those methods are calculated for each record, even though the values are never shown.

In Microsoft Dynamics AX 2009, Microsoft made a significant investment in the infrastructure of *cacheAddMethod*. In previous releases, this method worked only for display fields and only on form load. Beginning with Microsoft Dynamics AX 2009, the cache is used for both display and edit fields, and it is used throughout the lifetime of the form, including for reread, write, and refresh operations. It also works for any other method that reloads the data behind the form. With all of these methods, the fields are refreshed, but the kernel now refreshes them all at once instead of individually. In Microsoft Dynamics AX 2012, these features have been extended by another newly added feature—declarative display method caching.

Declarative display method caching

You can use the declarative display method caching feature to add a display method to the display method cache by setting the *CacheDataMethod* property on a form control to *Yes*. Figure 13-1 shows the *CacheDataMethod* property.

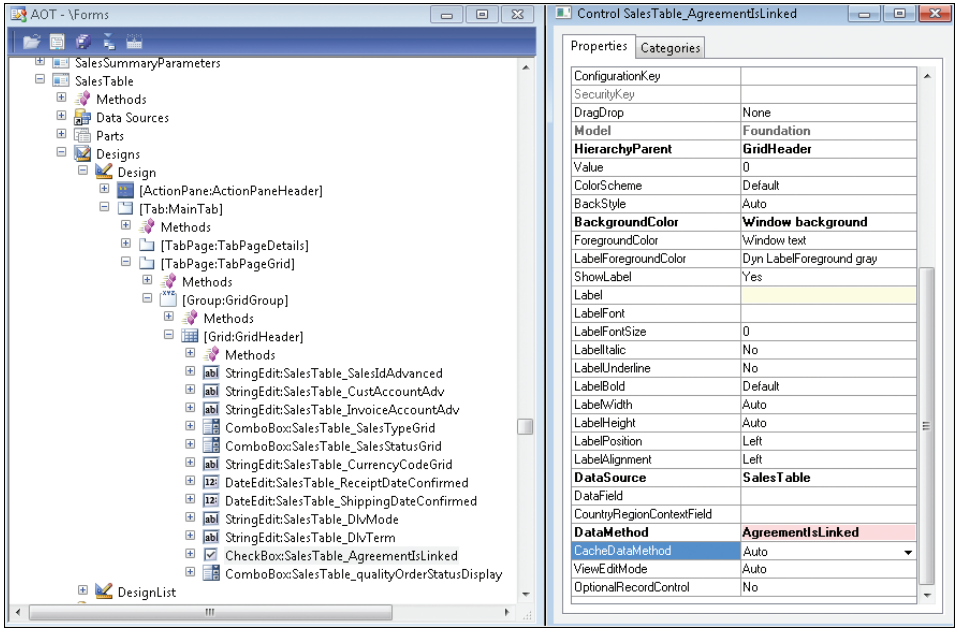


FIGURE 13-1 The *CacheDataMethod* property.

The values for the new property are *Auto*, *Yes*, and *No*, with the default value being *Auto*. *Auto* equates to *Yes* when the data method is hosted on a read-only form data source. This primarily applies to list pages. If the same data method is bound to multiple controls on a form, if at least one of them equates to *Yes*, the method is cached.

The RunBase technique

RunBase classes form the basis for most business logic in Microsoft Dynamics AX. *RunBase* provides much of the basic functionality necessary to execute a business process, such as displaying a dialog box, running the business logic, and running the business logic in batches.

Note Microsoft Dynamics AX 2012 introduces the SysOperation framework, which provides much of the functionality of the RunBase framework and will eventually replace it. For more information about the SysOperation framework in general, see Chapter 14, "Extending Microsoft Dynamics AX." For more information about optimizing performance when you use the SysOperation framework, see "The SysOperation framework," later in this chapter.

When business logic executes through the RunBase framework, the logic flows as shown in Figure 13-2.

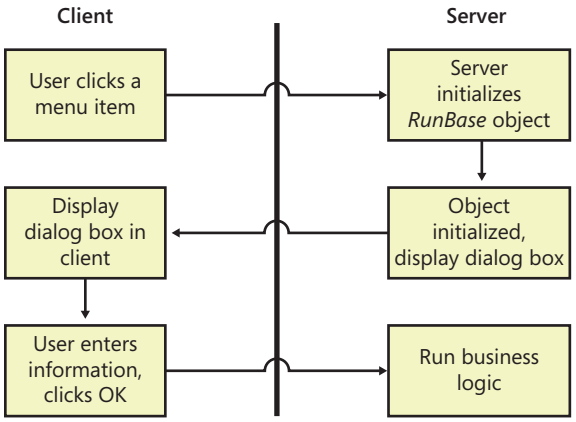


FIGURE 13-2 The RunBase communication pattern.

Most of the round-trip problems of the RunBase framework originate with the dialog box. For security reasons, the *RunBase* class should be running on the server because it accesses a large amount of data from the database and writes it back. But a problem occurs when the *RunBase* class is marked to run on the server. When the *RunBase* class runs on the server, the dialog box is created and driven from the server, causing excessive round-trips.

To avoid these round-trips, mark the *RunBase* class to run on *Called From*, meaning that it will run on either tier. Then mark either the *construct* method for the *RunBase* class or the menu item to run on the server. *Called From* enables the RunBase framework to marshal the class back and forth between the client and the server without having to drive the dialog box from the server, which significantly reduces the number of round-trips. Keep in mind that you must implement the *pack* and *unpack* methods in a way that allows this serialization to happen.

For an in-depth guide to implementing the RunBase framework to handle round-trips optimally between the client and the server, refer to the Microsoft Dynamics AX 2009 white paper, "RunBase Patterns," at *http://www.microsoft.com/en-us/download/details.aspx?id=19517*.

Caching and indexing

Microsoft Dynamics AX has a data caching framework on the client that can help you greatly reduce the number of times the client goes to the server. In Microsoft Dynamics AX, the cache operates across all of the unique keys in a table. Therefore, if a piece of code accesses data from the client, the code should use a unique key if possible. Also, you need to ensure that all unique keys are marked as such in the Application Object Tree (AOT). You can use the Best Practices tool to ensure that all of your tables have a primary key. For more information about the Best Practices tool, see Chapter 2, "The MorphX development environment and tools."

Setting the *CacheLookup* property correctly is a prerequisite for using the cache on the client. Table 13-1 shows the possible values for *CacheLookup*. These settings are discussed in greater detail in the "Caching" section later in this chapter.

TABLE 13-1 Settings for the *CacheLookup* property.

Cache setting	Description
Found	If a table is accessed through a primary key or a unique index, the value is cached for the duration of the session or until the record is updated. If another instance of the AOS updates this record, all AOS instances will flush their caches. This cache setting is appropriate for master data.
NotInTTS	Works the same way as *Found*, except that every time a transaction is started, the cache is flushed and the query goes to the database. This cache setting is appropriate for transactional tables.
FoundAndEmpty	Works the same way as *Found*, except that if the query cannot find a record, the absence of the record is stored. This cache setting is appropriate for region-specific master data or master data that isn't always present.
EntireTable	The entire table is cached in memory on the AOS, and the client treats this cache as *Found*. This cache setting is appropriate for tables with a known number of limited records, such as parameter tables.
None	No caching occurs. This setting is appropriate in only a few cases, such as when optimistic concurrency control must be disabled.

An index can be cached only if the *where* clause contains column names that are unique. The unique index join cache is a new feature that is discussed later in this chapter (see "The unique index join cache" in the "Transaction performance" section later in this chapter). This cache supports 1:1 relations only. In other words, caching won't work if a 1:*n* join is present or if the query is a cross-company query. In Microsoft Dynamics AX 2012, even if range operations are in the query, caching is supported so long as there is a unique key lookup in the query.

A cache that is set to *EntireTable* stores the entire contents of a table on the server, but the cache is treated as a *Found* cache on the client. For tables that have only one row for each company, such as parameter tables, add a key column that always has a known value, such as *0*. This allows the client to use the cache when accessing these tables. For an example of the use of a key column in Microsoft Dynamics AX, see the CustParameters table.

Write tier-aware code

When you're writing code, be aware of the tier that the code will run on and the tier that the objects you're accessing are on. Here are some things to be aware of:

- Objects whose *RunOn* property is set to *Server* are always instantiated on the server.
- Objects whose *RunOn* property is set to *Client* are always instantiated on the client.
- Objects whose *RunOn* property is set to *Called from* are instantiated wherever the class is created.

Note that if you mark classes to run on either the client or the server, you can't serialize them to another tier by using the *pack* and *unpack* methods. If you attempt to serialize a server class to the client, you get a new object on the server with the same values. Static methods run on whatever tier they are specified to run on by means of the *Client*, *Server*, or *Client Server* keyword in the declaration.

Handle *inMemory* temporary tables correctly

Temporary tables can be a common source of both client callbacks and calls to the server. Unlike regular table buffers, temporary tables are located on the tier on which the first record was inserted. For example, if a temporary table is declared on the server and the first record is inserted on the client, while the rest of the records are inserted on the server, all access to that table from the server happens on the client.

It's best to populate a temporary table on the server because the data that you need is probably coming from the database. Still, you must be careful when you want to iterate through the data to populate a form. The easiest way to achieve this efficiently is to populate the temporary table on the server, serialize the entire table to a container, and then read the records from the container into a temporary table on the client.

Avoid joining *inMemory* temporary tables with regular database tables whenever possible, because the AOS will first fetch all of the data in the database table of the current company and then combine the results in memory. This is an expensive, time-consuming process.

Try to avoid the type of code shown in the following example:

```
public static server void ImMemTempTableDemo()
{
    RealTable rt;
    InMemTempTable tt;
    int i;

    // Populate temp table
    ttsBegin;
    for (i=0; i<1000; i++)
    {
        tt.Value = int2str(i);
        tt.insert();
    }
    ttsCommit;
```

```
    // Inefficient join to database table. If the temporary table is an inMemory
    // temp table, this join causes 1,000 select statements on the database table and with
    // that, 1,000 round-trips to the database.

    select count(RecId) from tt join rt where tt.value == rt.Value;
    info(int642str(tt.Recid));
}
```

If you decide to use *inMemory* temporary tables, indexing them correctly for the queries that you plan to run on them will improve performance significantly. There is one difference compared to indexing for queries against regular tables: the fields must be in the same order as in the query itself. For example, the following query will benefit significantly from an index on the *AccountMain*, *ColumnId,* and *PeriodCode* fields in the TmpDimTransExtract table:

```
SELECT SUM(AmountMSTDebCred) FROM TmpDimTransExtract WHERE ((AccountMain>=N'11011201' AND
AccountMain<=N'11011299')) AND ((ColumnId = 1)) AND ((PeriodCode = 1))
```

Use *TempDB* temporary tables

You can use *TempDB* temporary tables to replace *inMemory* temporary table structures easily. *TempDB* temporary tables have the following advantages over *InMemory* temporary tables:

- You can join *TempDB* temporary tables to database tables efficiently.
- You can use set-based operations to populate *TempDB* temporary tables, reducing the number of round-trips to the database.

To create a *TempDB* temporary table, set the *TableType* property to *TempDB*, as shown in Figure 13-3.

FIGURE 13-3 Use the *TableType* property to create a *TempDB* temporary table.

Tip Even if temporary tables aren't dropped but are instead truncated and reused as soon as the current code goes out of scope, minimize the number of temporary tables that need to be created. There is a cost associated with creating a temporary table, so use them only if you need them.

If you use *TempDB* temporary tables, don't populate them by using line-based operations, as shown in the following example:

```
public static server void SQLTempTableDemo1()
{
```

```
    SQLTempTable tt;
    int i;

    // Populate temporary table; this will cause 1,000 round-trips to the database

    ttsBegin;
    for (i=0; i<1000; i++)
    {
        tt.Value = int2str(i);
        tt.insert();
    }
    ttsCommit;
}
```

Instead, use set-based operations. The following example shows how to use a set-based operation to create an efficient join to a database table:

```
public static server void SQLTempTableDemo2()
{
    RealTable rt;
    SQLTempTable tt;


    // Populate the temporary table with only one round-trip to the database.

    ttsBegin;
    insert_recordset tt (Value)
        select Value from rt;
    ttsCommit;

    // Efficient join to database table causes only one round-trip. If the temporary table
    // is an inMemory temp table, this join would cause 1,000 select statements on the
    // database table.

    select count(RecId) from tt join rt where tt.value == rt.Value;
    info(int642str(tt.Recid));
}
```

Eliminate client callbacks

A client callback occurs when the client places a call to a server-bound method and the server then places a call to a client-bound method. These calls can happen for two reasons. First, they occur if the client doesn't send enough information to the server during its call or if the client sends the server a client object that encapsulates the information. Second, they occur when the server is either updating or accessing a form.

To eliminate the first kind of callback, ensure that you send all of the information that the server needs in a serializable format, such as packed containers or value types (for example, *int*, *str*, *real*, or *boolean*). When the server accesses these types, it doesn't need to go back to the client the way that it does if you use an object type.

To eliminate the second type of callback, send any necessary information about the form to the method, and manipulate the form only when the call returns, instead of directly from the server. One of the best ways to defer operations on the client is by using the *pack* and *unpack* methods. With *pack* and *unpack*, you can serialize a class to a container and then deserialize it at the destination.

Group calls into chunks

To ensure the minimum number of round-trips between the client and the server, group calls into one static server method and pass in the state necessary to perform the operation.

The *NumberSeq::getNextNumForRefParmId* method is an example of a static server method that is used for this purpose. This method call contains the following line of code:

```
return NumberSeq::newGetNum(CompanyInfo::numRefParmId()).num();
```

If this code ran on the client, it would cause four remote procedure call (RPC) round-trips: one for *newGetNum*, one for *numRefParmId*, one for *num*, and one to clean up the *NumberSeq* object that was created. By using a static server method, you can complete this operation in one RPC round-trip.

Another common example of grouping calls into chunks occurs when the client performs transaction tracking system (TTS) operations. Frequently, a developer writes code similar to that in the following example:

```
ttsBegin;
record.update();
ttsCommit;
```

You can save two round-trips if you group this code into one static server call. All TTS operations are initiated only on the server. To take advantage of this, do not invoke the *ttsbegin* and *ttscommit* call from the client to start the database transaction when the *ttslevel* is *0*.

Pass table buffers by value instead of by reference

The global methods *buf2con* and *con2buf* are used in X++ to convert table buffers into containers, and vice versa. New functionality has been added to these methods, and they have been improved to run much faster than in previous versions of Microsoft Dynamics AX.

Converting table buffers into containers is useful if you need to send the table buffer across different tiers; (for example, between the client and the server). Sending a container is better than sending a table buffer because containers are passed by value and table buffers are passed by reference. Passing objects by reference across tiers causes a high number of RPC calls and degrades the performance of your application. Referencing objects that were created on different tiers causes an RPC call every time the other tier invokes one of the instance methods of the remote object. To improve performance, you can eliminate a callback by creating local copies of the table buffers, using *buf2con* to pack the table and *con2buf* to unpack it.

The following example shows a form running on the client and transferring data to the server for updating. The example illustrates how to transfer a buffer efficiently with a minimum number of RPC calls:

Note In practice, you would not use a temporary table and would access actual database data.

```
public void updateResultField(Buf2conExample   clientRecord)
{
    container   packedRecord;

    // Pack the record before sending to the server

    packedRecord = buf2Con(clientRecord);

    // Send packed record to the server and container with the result

    packedRecord = Buf2ConExampleServerClass::modifyResultFromPackedRecord(packedRecord);

    // Unpack the returned container into the client record.

    con2Buf(packedRecord, clientRecord);
    Buf2conExample_ds.refresh();
}
```

Modify the data on the server tier and then send a container back:

```
public static server container  modifyResultFromPackedRecord(container _packedRecord)
{
    Buf2conExample recordServerCopy = con2Buf(_packedRecord);
    Buf2ConExampleServerClass::modifyResult(recordServerCopy);
    return buf2Con(recordServerCopy);
}
public static server void modifyResult(Buf2conExample  _clientTmpRecord)
{
    int n = _clientTmpRecord.A;
    _clientTmpRecord.Result = 0;
    while (n > 0)
    {
        _clientTmpRecord.Result = Buf2ConExampleServerClass::add(_clientTmpRecord);
        n--;
    }
}
```

Transaction performance

The preceding section focused on limiting traffic between the client and server tiers. When a Microsoft Dynamics AX application runs, however, these are just two of the three tiers that are involved. The third tier is the database tier. You must optimize the exchange of packages between the server tier and the database tier, just as you do between the client tier and the server tier. This section explains how you can optimize transactions.

The Microsoft Dynamics AX run time helps you minimize calls made from the server tier to the database tier by supporting set-based operators and data caching. However, you should also do your part by reducing the amount of data you send from the database tier to the server tier. The less data you send, the faster that data is retrieved from the database and fewer packages are sent back. These reductions result in less memory being consumed. All of these efforts promote faster execution of application logic, which results in smaller transaction scope, less locking and blocking, and improved concurrency and throughput.

Note You can improve transaction performance further through the design of your application logic. For example, ensuring that various tables and records are always modified in the same order helps prevent deadlocks and ensuing retries. Spending time preparing the transactions to be as brief as possible before starting a transaction scope can reduce the locking scope and resulting blocking, ultimately improving the concurrency of the transactions. Database design factors, such as index design and use, are also important. However, these topics are beyond the scope of this book.

Set-based data manipulation operators

The X++ language contains operators and classes to enable set-based manipulation of the database. Set-based constructs have an advantage over record-based constructs—they make fewer round-trips to the database. The following X++ code example, which selects several records in the CustTable table and updates each record with a new value in the *CreditMax* field, illustrates how a round-trip is required when the *select* statement executes and each time the *update* statement executes:

```
static void UpdateCustomers(Args _args)
{
    CustTable custTable;

    ttsBegin;

    while select forupdate custTable
        where custTable.CustGroup == '20' // Round-trips to the database
    {
        custTable.CreditMax = 1000;
        custTable.update(); // Round-trip to the database
    }

    ttsCommit;
}
```

In a scenario in which 100 CustTable records qualify for the update because the *CustGroup* field value equals *20*, the number of round-trips would be 101 (1 for the *select* statement and 100 for the *update* statements). The number of round-trips for the *select* statement might actually be slightly higher, depending on the number of CustTable records that can be retrieved simultaneously from the database and sent to the AOS.

Theoretically, you could rewrite the code in the preceding example to result in only one round-trip to the database by changing the X++ code, as indicated in the following example. This example shows how to use the set-based *update_recordset* operator, resulting in a single Transact-SQL *UPDATE* statement being passed to the database:

```
static void UpdateCustomers(Args _args)
{
    CustTable custTable;

    ttsBegin;

    update_recordset custTable setting CreditMax = 1000
        where custTable.CustGroup == '20'; // Single round-trip to the database

    ttsCommit;
}
```

For several reasons, however, using a record buffer for the CustTable table doesn't result in only one round-trip. The reasons are explained in the following sections about the set-based constructs that the Microsoft Dynamics AX run time supports. These sections also describe features that you can use to ensure a single round-trip to the database, even when you're using a record buffer for the table.

Important The set-based operations described in the following sections do not improve performance when used on *inMemory* temporary tables. The Microsoft Dynamics AX run time always downgrades set-based operations on *inMemory* temporary tables to record-based operations. This downgrade happens regardless of how the table became a temporary table (whether specified in metadata in the table's properties, disabled because of the configuration of the Microsoft Dynamics AX application, or explicitly stated in the X++ code that references the table). Also, the downgrade always invokes the *doInsert*, *doUpdate*, and *doDelete* methods on the record buffer, so no application logic in the overridden methods is executed.

Set-based operations and table hierarchies

A set-based operation such as *insert_recordset*, *update_recordset*, or *delete_from* is not downgraded to a record-based operation on a subtype or supertype table unless a condition that would cause the operation to be downgraded is met. Both an *insert_recordset* and *update_recordset* can update or insert all qualifying records into the specified table and all subtype and supertype tables, but not into any derived tables. The *delete_from* operator is treated differently because it deletes all qualifying records from the current table and its subtype and supertype tables to guarantee that the record is deleted completely from the database. For more information about the conditions that cause a downgrade, review the following sections.

The *insert_recordset* operator

The *insert_recordset* operator enables the insertion of multiple records into a table in one round-trip to the database. The following X++ code illustrates the use of *insert_recordset*. The code copies entries for one item in the InventTable table and the InventSum table into a temporary table for future use:

```
static void CopyItemInfo(Args _args)
{
    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    // insert_recordset uses only one round-trip for the copy operation.
    // A record-based insert would need one round-trip per record in InventSum.

    ttsBegin;
    insert_recordset insertInventTableInventSum (ItemId,AltItemId,PhysicalValue,PostedValue)
        select ItemId,AltItemid from inventTable where inventTable.ItemId == '1001'
        join PhysicalValue,PostedValue from inventSum
        where inventSum.ItemId == inventTable.ItemId;
    ttsCommit;
    elect count(RecId) from insertInventTableInventSum;
    info(int642str(insertInventTableInventSum.RecId));

    // Additional code to use the copied data.
}
```

The round-trip to the database involves the execution of three statements in the database:

1. The *select* part of the *insert_recordset* statement executes when the selected rows are inserted into a new temporary table in the database. The syntax of the *select* statement when executed in Transact-SQL is similar to *SELECT <field list> INTO <temporary table> FROM <source tables> WHERE <predicates>*.
2. The records from the temporary table are inserted directly into the target table using syntax such as *INSERT INTO <target table> (<field list>) SELECT <field list> FROM <temporary table>*.
3. The temporary table is dropped with the execution of *DROP TABLE <temporary table>*.

This approach has a tremendous performance advantage over inserting the records one by one, as shown in the following X++ code, which addresses the same scenario:

```
static void CopyItemInfoLineBased(Args _args)
{
    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    ttsBegin;
    while select ItemId,Altitemid from inventTable where inventTable.ItemId == '1001'
        join PhysicalValue,PostedValue from inventSum
        where inventSum.ItemId == inventTable.ItemId
```

```
    {
        InsertInventTableInventSum.ItemId        = inventTable.ItemId;
        InsertInventTableInventSum.AltItemId     = inventTable.AltItemId;
        InsertInventTableInventSum.PhysicalValue = inventSum.PhysicalValue;
        InsertInventTableInventSum.PostedValue   = inventSum.PostedValue;
        InsertInventTableInventSum.insert();
    }
    ttsCommit;

    select count(RecId) from insertInventTableInventSum;
    info(int642str(insertInventTableInventSum.RecId));

    // ... Additional code to use the copied data
}
```

If the InventSum table contains 10 entries for which *ItemId* equals *1001*, this scenario would result in one round-trip for the *select* statement and an additional 10 round-trips for the inserts, totaling 11 round-trips.

The *insert_recordset* operation can be downgraded from a set-based operation to a record-based operation if any of the following conditions is true:

- The table is cached by using the *EntireTable* setting.
- The *insert* method or the *aosValidateInsert* method is overridden on the target table.
- Alerts are set to be triggered by inserts into the target table.
- The database log is configured to log inserts into the target table.
- Record-level security (RLS) is enabled on the target table. If RLS is enabled only on the source table or tables, *insert_recordset* isn't downgraded to a row-by-row operation.
- The *ValidTimeStateFieldType* property for a table is not set to *None*.

The Microsoft Dynamics AX run time automatically handles the downgrade and internally executes a scenario similar to the *while select* scenario shown in the preceding example.

Important When the Microsoft Dynamics AX run time checks for overridden methods, it determines only whether the methods are implemented. It doesn't determine whether the overridden methods contain only the default X++ code. A method is therefore considered to be overridden by the run time even though it contains the following X++ code:

```
public void insert()
{
    super();
}
```

Any set-based insert is then downgraded.

Unless a table is cached by using the *EntireTable* setting, you can avoid the downgrade caused by the other conditions mentioned earlier. The record buffer contains methods that turn off the checks that the run time performs when determining whether to downgrade the *insert_recordset* operation:

- Calling *skipDataMethods(true)* prevents the check that determines whether the *insert* method is overridden.
- Calling *skipAosValidation(true)* prevents the check on the *aosValidateInsert* method.
- Calling *skipDatabaseLog(true)* prevents the check that determines whether the database log is configured to log inserts into the table.
- Calling *skipEvents(true)* prevents the check that determines whether any alerts have been set to be triggered by the *insert* event on the table.

The following X++ code, which includes the call to *skipDataMethods(true)*, ensures that the *insert_recordset* operation is not downgraded because the *insert* method is overridden on the InventSize table:

```
static void CopyItemInfoskipDataMethod(Args _args)
{
    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    ttsBegin;

    // Skip override check on insert.

    insertInventTableInventSum.skipDataMethods(true);
    insert_recordset insertInventTableInventSum (ItemId,AltItemId,PhysicalValue,PostedValue)
        select ItemId,Altitemid from inventTable where inventTable.ItemId == '1001'
        join PhysicalValue,PostedValue from inventSum
        where inventSum.ItemId == inventTable.ItemId;
    ttsCommit;

    select count(RecId) from insertInventTableInventSum;
    info(int642str(insertInventTableInventSum.RecId));

    // ... Additional code to use the copied data
}
```

Important Use the *skip* methods with extreme caution because they can prevent the logic in the *insert* method from being executed, prevent events from being raised, and potentially, prevent the database log from being written to.

If you override the *insert* method, use the cross-reference system to determine whether any X++ code calls *skipDataMethods(true)*. If you don't, the X++ code might fail to execute the *insert* method. Moreover, when you implement calls to *skipDataMethods(true)*, ensure that data inconsistency will not result if the X++ code in the overridden *insert* method doesn't execute.

You can use *skip* methods only to influence whether the *insert_recordset* operation is downgraded. If you call *skipDataMethods(true)* to prevent a downgrade because the *insert* method is overridden, use the Microsoft Dynamics AX Trace Parser to make sure that the operation has not been downgraded. The operation is downgraded if, for example, the database log is configured to log inserts into the table. In the previous example, the overridden *insert* method on the InventSize table would be executed if the database log were configured to log inserts into the InventSize table, because the *insert_recordset* operation would then revert to a *while select* scenario in which the overridden *insert* method would be called. For more information about the Trace Parser, see the section "Performance monitoring tools" later in this chapter.

Since the Microsoft Dynamics AX 2009 release, the *insert_recordset* operator has supported literals. Support for literals was introduced primarily to support upgrade scenarios in which the target table is populated with records from one or more source tables (using joins) and one or more columns in the target table must be populated with a literal value that doesn't exist in the source. The following code example illustrates the use of literals in *insert_recordset*:

```
static void CopyItemInfoLiteralSample(Args _args)
{
    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;
    boolean                    flag = boolean::true;

    ttsBegin;
    insert_recordset insertInventTableInventSum
(ItemId,AltItemId,PhysicalValue,PostedValue,Flag)
        select ItemId,altitemid from inventTable where inventTable.ItemId == '1001'
        join PhysicalValue,PostedValue,Flag from inventSum
        where inventSum.ItemId == inventTable.ItemId;
    ttsCommit;

    select firstonly ItemId,Flag from insertInventTableInventSum;
    info(strFmt('%1,%2',insertInventTableInventSum.ItemId,insertInventTableInventSum.Flag));
    // ... Additional code to utilize the copied data
}
```

The *update_recordset* operator

The behavior of the *update_recordset* operator is similar to that of the *insert_recordset* operator. This similarity is illustrated by the following piece of X++ code, in which all rows that have been inserted for one *ItemId* are updated and flagged for further processing:

```
static void UpdateCopiedData(Args _args)
{

    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    // Code assumes InsertInventTableInventSum is populated.
```

```
    // Set-based update operation.
    ttsBegin;
    update_recordSet insertInventTableInventSum setting Flag = true
    where insertInventTableInventSum.ItemId == '1001';
    ttsCommit;
}
```

The execution of *update_recordset* results in one statement being passed to the database—which in Transact-SQL uses syntax similar to *UPDATE <table> <SET> <field and expression list> WHERE <predicates>*. As with *insert_recordset*, *update_recordset* provides a tremendous performance improvement over the record-based version that updates each record individually. This improvement is shown in the following X++ code, which serves the same purpose as the preceding example. The code selects all of the records that qualify for update, sets the new description value, and updates the record:

```
static void UpdateCopiedDataLineBased(Args _args)
{

    InventTable               inventTable;
    InventSum                 inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    // ... Code assumes InsertInventTableInventSum is populated

    ttsBegin;
    while select forUpdate InsertInventTableInventSum
    where insertInventTableInventSum.ItemId == '1001'
    {
        insertInventTableInventSum.Flag = true;
        insertInventTableInventSum.update();
    }
    ttsCommit;
}
```

If ten records qualify for the update, one *select* statement and ten *update* statements are passed to the database, rather than the single *update* statement that would be passed with *update_recordset*.

The *update_recordset* operation can be downgraded if specific methods are overridden or if Microsoft Dynamics AX is configured in specific ways. The *update_recordset* operation is downgraded if any of the following conditions is true:

- The table is cached by using the *EntireTable* setting.
- The *update* method, the *aosValidateUpdate* method, or the *aosValidateRead* method is overridden on the target table.
- Alerts are set up to be triggered by *update* queries on the target table.
- The database log is configured to log *update* queries on the target table.
- RLS is enabled on the target table.
- The *ValidTimeStateFieldType* property for a table is not set to *None*.

The Microsoft Dynamics AX run time automatically handles the downgrade and internally executes a scenario similar to the *while select* scenario shown in the earlier example.

As with the *insert_recordset* operator, you can avoid a downgrade unless the table is cached by using the *EntireTable* setting. The record buffer contains methods that turn off the checks that the run time performs when determining whether to downgrade the *update_recordset* operation:

- Calling *skipDataMethods(true)* prevents the check that determines whether the *update* method is overridden.
- Calling *skipAosValidation(true)* prevents the checks on the *aosValidateUpdate* and *aosValidateRead* methods.
- Calling *skipDatabaseLog(true)* prevents the check that determines whether the database log is configured to log updates to records in the table.
- Calling *skipEvents(true)* prevents the check to determine whether any alerts have been set to be triggered by the *update* event on the table.

As explained earlier, use the *skip* methods with caution. Again, using the *skip* methods only influences whether the *update_recordset* operation is downgraded to a *while select* operation. If the operation is downgraded, database logging, alerting, and execution of overridden methods occur even though the respective *skip* methods have been called.

Tip If an *update_recordset* operation is downgraded, the *select* statement uses the concurrency model specified at the table level. You can apply the *optimisticlock* and *pessimisticlock* keywords to the *update_recordset* statements and enforce a specific concurrency model to be used in case of a downgrade.

Microsoft Dynamics AX supports inner and outer joins in *update_recordset*. The support for joins in *update_recordset* enables an application to perform set-based operations when the source data is fetched from more than one related data source.

The following example illustrates the use of joins with *update_recordset*:

```
static void UpdateCopiedDataJoin(Args _args)
{
    InventTable               inventTable;
    InventSum                 inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    // ... Code assumes InsertInventTableInventSum is populated
    // Set-based update operation with join.

    ttsBegin;
    update_recordSet insertInventTableInventSum setting Flag = true,
    DiffAvailOrderedPhysical = inventSum.AvailOrdered - inventSum.AvailPhysical
```

```
    join InventSum where inventSum.ItemId == insertInventTableInventSum.ItemId &&
    inventSum.AvailOrdered > inventSum.AvailPhysical;
    ttsCommit;
}
```

The *delete_from* operator

The *delete_from* operator is similar to the *insert_recordset* and *update_recordset* operators in that it passes a single statement to the database to delete multiple rows, as shown in the following code:

```
static void DeleteCopiedData(Args _args)
{
    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    // ... Code assumes InsertInventTableInventSum is populated
    // Set-based delete operation

    ttsBegin;
    delete_from insertInventTableInventSum
    where insertInventTableInventSum.ItemId == '1001';
    ttsCommit;
}
```

This code passes a statement to Microsoft SQL Server in a syntax similar to *DELETE <table> WHERE <predicates>* and performs the same actions as the following X++ code, which uses record-by-record deletes:

```
static void DeleteCopiedDataLineBased(Args _args)
{

    InventTable                inventTable;
    InventSum                  inventSum;
    InsertInventTableInventSum insertInventTableInventSum;

    // ... Code assumes InsertInventTableInventSum is populated

    ttsBegin;
    while select forUpdate insertInventTableInventSum
    where insertInventTableInventSum.ItemId == '1001'
    {
        insertInventTableInventSum.delete();
    }
    ttsCommit;
}
```

Again, the use of *delete_from* is preferable for performance because a single statement is passed to the database, instead of the multiple statements that the record-by-record version parses.

As with the *insert_recordset* and *update_recordset* operations, the *delete_from* operation can be downgraded—and for similar reasons. A downgrade occurs if any of the following conditions is true:

- The table is cached by using the *EntireTable* setting.
- The *delete* method, the *aosValidateDelete* method, or the *aosValidateRead* method is overridden on the target table.
- Alerts are set up to be triggered by deletions from the target table.
- The database log is configured to log deletions from the target table.
- The *ValidTimeStateFieldType* property for a table is not set to *None*.

A downgrade also occurs if delete actions are defined on the table. The Microsoft Dynamics AX run time automatically handles the downgrade and internally executes a scenario similar to the *while select* operation shown in the earlier example.

You can avoid a downgrade caused by these conditions unless the table is cached by using the *EntireTable* setting. The record buffer contains methods that turn off the checks that the run time performs when determining whether to downgrade the *delete_from* operation, as follows:

- Calling *skipDataMethods(true)* prevents the check that determines whether the *delete* method is overridden.
- Calling *skipAosValidation(true)* prevents the checks on the *aosValidateDelete* and *aosValidateRead* methods.
- Calling *skipDatabaseLog(true)* prevents the check that determines whether the database log is configured to log the deletion of records in the table.
- Calling *skipEvents(true)* prevents the check that determines whether any alerts have been set to be triggered by the *delete* event on the table.

The preceding descriptions about the use of the *skip* methods, the no-skipping behavior in the event of downgrade, and the concurrency model for the *update_recordset* operator are equally valid for the use of the *delete_from* operator.

Note The record buffer also contains a *skipDeleteMethod* method. Calling the method as *skipDeleteMethod(true)* has the same effect as calling *skipDataMethods(true)*. It invokes the same Microsoft Dynamics AX run-time logic, so you can use *skipDeleteMethod* in combination with *insert_recordset* and *update_recordset*, although it might not improve the readability of the X++ code.

The *RecordInsertList* and *RecordSortedList* classes

In addition to the set-based operators, you can use the *RecordInsertList* and *RecordSortedList* classes when inserting multiple records into a table. When the records are ready to be inserted, the Microsoft Dynamics AX run time packs multiple records into a single package and sends it to the database. The database then executes an individual insert operation for each record in the package. This process is illustrated in the following example, in which a *RecordInsertList* object is instantiated, and each record to be inserted into the database is added to the *RecordInsertList* object. When all records are inserted into the object, the *insertDatabase* method is called to ensure that all records are inserted into the database.

```
static void CopyItemInfoRIL(Args _args)
{
    InventTable                     inventTable;
    InventSum                       inventSum;
    InsertInventTableInventSumRT    insertInventTableInventSumRT;
    RecordInsertList                ril;

    ttsBegin;
    ril = new RecordInsertList(tableNum(InsertInventTableInventSumRT));

    while select ItemId,AltItemid from inventTable where inventTable.ItemId == '1001'
    join PhysicalValue,PostedValue from inventSum
    where inventSum.ItemId == inventTable.ItemId
    {
        insertInventTableInventSumRT.ItemId        = inventTable.ItemId;
        insertInventTableInventSumRT.AltItemId     = inventTable.AltItemId;
        insertInventTableInventSumRT.PhysicalValue = inventSum.PhysicalValue;
        insertInventTableInventSumRT.PostedValue   = inventSum.PostedValue;
        // Insert records if package is full
        ril.add(insertInventTableInventSumRT);
    }

    // Insert remaining records into database

    ril.insertDatabase();
    ttsCommit;

    select count(RecId) from insertInventTableInventSumRT;
    info(int642str(insertInventTableInventSumRT.RecId));

    // Additional code to use the copied data.
}
```

Based on the maximum buffer size that is configured for the server, the Microsoft Dynamics AX run time determines the number of records in a buffer as a function of the size of the records and the buffer size. If the buffer is full, the records in the *RecordInsertList* object are packed, passed to the database, and inserted individually on the database tier. This check is made when the *add* method is called. When the *insertDatabase* method is called from application logic, the remaining records are inserted with the same mechanism.

Using these classes has an advantage over using *while select*: fewer round-trips are made from the AOS to the database because multiple records are sent simultaneously. However, the number of *INSERT* statements in the database remains the same.

Note Because the timing of insertion into the database depends on the size of the record buffer and the package, don't expect a record to be selectable from the database until the *insertDatabase* method has been called.

You can rewrite the preceding example by using the *RecordSortedList* class instead of *RecordInsertList*, as shown in the following X++ code:

```
public static server void CopyItemInfoRSL()
{
    InventTable                 inventTable;
    InventSum                   inventSum;
    InsertInventTableInventSumRT insertInventTableInventSumRT;
    RecordSortedList            rsl;

    ttsBegin;
    rsl = new RecordSortedList(tableNum(InsertInventTableInventSumRT));
    rsl.sortOrder(fieldNum(InsertInventTableInventSumRT,PostedValue));

    while select ItemId,AltItemid from inventTable where inventTable.ItemId == '1001'
    join PhysicalValue,PostedValue from inventSum
    where inventSum.ItemId == inventTable.ItemId
    {
        insertInventTableInventSumRT.ItemId        = inventTable.itemId;
        insertInventTableInventSumRT.AltItemId     = inventTable.AltItemId;
        insertInventTableInventSumRT.PhysicalValue = inventSum.PhysicalValue;
        insertInventTableInventSumRT.PostedValue   = inventSum.PostedValue;

        //No records will be inserted.
        rsl.ins(insertInventTableInventSumRT);
    }

    //All records are inserted in database.
    rsl.insertDatabase();
    ttsCommit;

    select count(RecId) from insertInventTableInventSumRT;
    info(int642str(insertInventTableInventSumRT.RecId));

    // Additional code to utilize the copied data
}
```

When the application logic uses a *RecordSortedList* object, the records aren't passed and inserted in the database until the *insertDatabase* method is called. The number of round-trips and *INSERT* statements executed is the same as for the *RecordInsertList* object.

Both *RecordInsertList* objects and *RecordSortedList* objects can be downgraded in application logic to record-by-record inserts, in which each record is sent in a separate round-trip to the database

and the *INSERT* statement is subsequently executed. A downgrade occurs if the *insert* method or the *aosValidateInsert* method is overridden, or if the table contains fields of the type *container* or *memo*. However, no downgrade occurs if the database log is configured to log inserts or alerts that are set to be triggered by the *insert* event on the table. One exception is if logging or alerts have been configured and the table contains *CreatedDateTime* or *ModifiedDateTime* columns—in this case, record-by-record inserts are performed. The database logging and alerts occur on a record-by-record basis after the records have been sent and inserted into the database.

When instantiating the *RecordInsertList* object, you can specify that the *insert* and *aosValidateInsert* methods be skipped. You can also specify that the database logging and eventing be skipped if the operation isn't downgraded.

Tips for transferring code into set-based operations

Often, code is not transferred to a set-based operation because the logic is too complex. However, an *if* condition, for example, can be placed in the *where* clause of a query. If you have a scenario that requires an *if/else* decision, you can achieve this with two queries, such as two *update_recordsets*. Necessary information from other tables can be obtained through joins instead of being looked up in a *find* operation. In Microsoft Dynamics AX 2012 *insert_recordset* and *TempDB* temporary tables help to extend the possibilities of transferring code into set-based operations.

Some things still might seem difficult to transfer to a set-based operation, such as performing calculations on the columns in a *select* statement. For this reason, Microsoft Dynamics AX 2012 offers a feature for views that is called *computed columns*, and you can use this feature to transfer even fairly complex logic into set-based operations. Computed columns can also provide performance advantages when used as an alternative to display methods on read-only data sources. Imagine the following task: Find all customers who bought products for more than $100,000 and all customers who bought products for more than $1,000,000. Those customers are treated as VIP customers who then get certain rebates.

In earlier versions of Microsoft Dynamics AX, the X++ code to set these values would have looked like the following example:

```
public static server void demoOld()
{
    SalesLine  sl;
    CustTable  ct;
    vipparm    vp;
    int64      total;

    vp = vipparm::find();
    ttsBegin;

    // One + n round-trips per Customer Account in the salesline table.
    while select CustAccount, sum(SalesQty), sum(SalesPrice) from sl group by sl.CustAccount
    {

    // Necessary to select for update causing n additional round-trips.
    ct = CustTable::find(sl.CustAccount,true);
```

```
    ct.VIPStatus = 0;

        if((sl.SalesQty*sl.SalesPrice)>=vp.UltimateVIP)
            ct.VIPStatus = 2;
        else if((sl.SalesQty*sl.SalesPrice)>=vp.VIP)
            ct.VIPStatus = 1;

     // Another n round-trips for the update.
        if(ct.VIPStatus != 0)
            ct.update();
    }
    ttsCommit;
}
```

You could replace this code easily with two direct Transact-SQL statements to make it far more effective. The direct Transact-SQL statements would look like the following:

```
UPDATE CUSTTABLE SET VIPSTATUS = 2 FROM (SELECT CUSTACCOUNT,SUM(SALESQTY)*SUM(SALESPRICE) AS
TOTAL,VIPSSTATUS = CASE
 WHEN SUM(SALESQTY)*SUM(SALESPRICE) > 1000000 THEN 2
 WHEN SUM(SALESQTY)*SUM(SALESPRICE) > 100000  THEN 1
 ELSE 0 END
 FROM SALESLINE GROUP BY CUSTACCOUNT) AS VC WHERE VC.VIPSTATUS = 2 and CUSTTABLE.ACCOUNTNUM =
VC.CUSTACCOUNT and DATAAREAID = N'CEU'
```

Note This code contains only a partial *dataAreaId* and no *Partition* field, which highlights its weaknesses. The data access logic is not enforced.

In Microsoft Dynamics AX 2012, with the help of computed columns, you can replace this code with two set-based statements.

To create these statements, you first need to create an AOT query because views themselves cannot contain a *group by* statement. Further, you need a parameter table that holds the information about who counts as a VIP customer for each company (Figure 13-4). Then, you need to join this information together so that it is available at run time.

The code for the computed column is shown here:

```
private static server str compColQtyPrice()
{
    str sReturn,sQty,sPrice,ultimateVIP,VIP;
    Map m = new Map(Types::String,Types::String);
    sQty        = SysComputedColumn::returnField(tableStr(mySalesLineView),
                                                 identifierStr(SalesLine_1),
                                                 fieldStr(SalesLine,SalesQty));
    sPrice      = SysComputedColumn::returnField(tableStr(mySalesLineView),
                                                 identifierStr(SalesLine_1),
                                                 fieldStr(SalesLine,SalesPrice));
    ultimateVIP = SysComputedColumn::returnField(tableStr(mySalesLineView),
                                                 identifierStr(Vipparm_1),
                                                 fieldStr(vipparm,ultimateVIP));
```

```
    VIP         = SysComputedColumn::returnField(tableStr(mySalesLineView),
                                                 identifierStr(Vipparm_1),
                                                 fieldStr(vipparm,VIP));
    m.insert(SysComputedColumn::sum(sQty)+'*'+SysComputedColumn::sum(sPrice)+
             ' > '+ultimateVIP,int2str(VipStatus::UltimateVIP));
    m.insert(SysComputedColumn::sum(sQty)+'*'+SysComputedColumn::sum(sPrice)+
             ' > '+VIP ,int2str(VipStatus::VIP));
    return SysComputedColumn::switch('',m,'0');
}
```

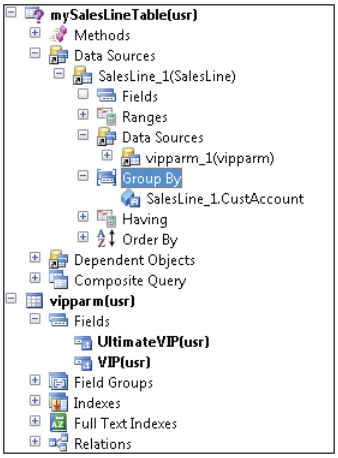


FIGURE 13-4 Creating the parameter table and the initial query.

The next step is to add the parameter table to a view and create the necessary computed column, as shown in Figure 13-5.

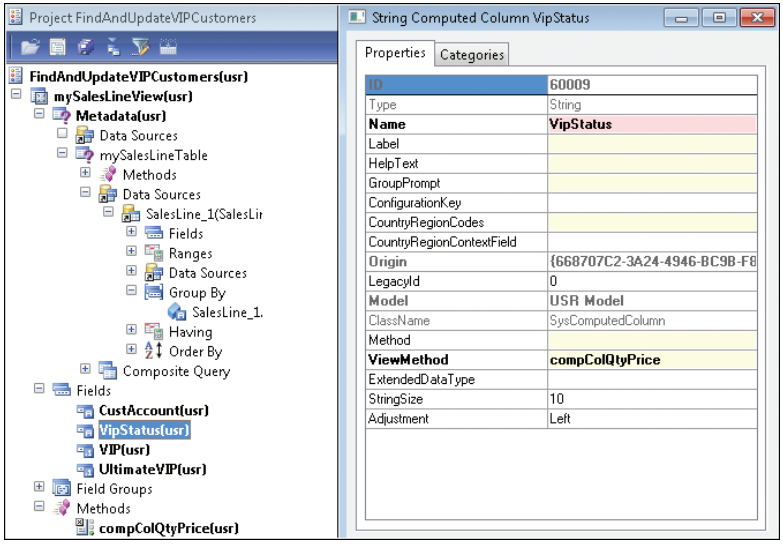


FIGURE 13-5 Creating the view and the computed column.

The view in SQL Server looks like this:

```
SELECT T1.CUSTACCOUNT AS CUSTACCOUNT,T1.DATAAREAID AS DATAAREAID,1010 AS RECID,T2.DATAAREAID
AS DATAAREAID#2,T2.VIP AS VIP,T2.ULTIMATEVIP AS ULTIMATEVIP,(CAST ((CASE  WHEN SUM(T1.
SALESQTY)*SUM(T1.SALESPRICE) > T2.ULTIMATEVIP THEN 2 WHEN SUM(T1.SALESQTY)*SUM(T1.SALESPRICE) >
T2.VIP THEN 1 ELSE 0 END) AS NVARCHAR(10))) AS VIPSTATUS FROM SALESLINE T1 CROSS JOIN VIPPARM T2
GROUP BY T1.CUSTACCOUNT,T1.DATAAREAID,T2.DATAAREAID,T2.VIP,T2.ULTIMATEVIP
```

Now you can change the record-based update code used earlier to effective, working set-based code:

```
public static server void demoNew()
{
    mySalesLineView mySLV;
    CustTable       ct;
    ct.skipDataMethods(true);
    update_recordSet ct setting VipStatus = VipStatus::UltimateVIP
    join mySLV where ct.AccountNum == mySLV.CustAccount &&
    mySLV.VipStatus == int2str(enum2int(vipstatus::UltimateVIP));
    update_recordSet ct setting VipStatus = VipStatus::VIP
    join mySLV where ct.AccountNum == mySLV.CustAccount &&
    mySLV.VipStatus == int2str(enum2int(vipstatus::VIP));
}
```

Executing the code shows the difference in timing:

```
public static void main(Args _args)
{
    int tickcnt;
    DemoClass::resetCusttable();
    tickcnt = WinAPI::getTickCount();
    DemoClass::demoOld();
    info('Line based' + int2str(WinAPI::getTickCount()-tickcnt));
    DemoClass::resetCusttable();
    tickcnt = WinAPI::getTickCount();
    DemoClass::demoNew();
    info('Set based' + int2str(WinAPI::getTickCount()-tickcnt));
}
```

The execution time of the operation is as follows:

- **Record-based** 1,514 milliseconds
- **Set-based** 171 milliseconds

Note that this code ran on demo data. Imagine running similar code on an actual database with hundreds of thousands of sales orders and customers.

Another example that might seem tricky to transfer to a set-based operation is if you need to use aggregation and *group by* in queries, because since the *update_recordset* operator does not support this. You can work around this issue by using *TempDB* temporary tables and a combination of *insert_recordset* and *update_recordset*.

Note The amount of data that you need to modify determines whether this pattern is beneficial. For example, if you just want to update 10 rows, a *while select* statement might be more efficient. But if you are updating hundreds or thousands of rows, this pattern can be more efficient. You'll need to evaluate and test each pattern individually to determine which one provides better performance.

The following example first populates a table and then updates the values in it based on a *group by* and *sum* operations in a statement. Note that deleting and populating the data takes longer than the actual execution of the later *insert_recordset* and *update_recordset* statements.

```
public static server void PopulateTable()
{
    MyUpdRecordsetTestTable MyUpdRecordsetTestTable;
    int myGrouping,myKey,mySum;
    RecordInsertList   ril = new RecordInsertList(tablenum(MyUpdRecordsetTestTable));

    delete_from MyUpdRecordsetTestTable;

    for(myKey=0;myKey<=100000;myKey++)
    {
        MyUpdRecordsetTestTable.Key     = myKey;
        if(myKey mod 10 == 0)
        {
          myGrouping += 10;
          mySum     += 10;
        }
        MyUpdRecordsetTestTable.fieldForGrouping    = myGrouping;
        MyUpdRecordsetTestTable.theSum              = mySum;
        ril.add(MyUpdRecordsetTestTable);
    }
    ril.insertDatabase();
}
```

Combine *TempDB* temporary tables, *insert_recordset,* and *update_recordset* to update the table:

```
public static void InsertAndUpdate()
{
    MyUpdRecordsetTestTable MyUpdRecordsetTestTable;
    MyUpdRecordsetTestTableTmp MyUpdRecordsetTestTableTmp;
    int tc;

    tc = WinAPI::getTickCount();
    insert_recordset MyUpdRecordsetTestTableTmp(fieldForGrouping,theSum)
    select fieldForGrouping,sum(theSum) from MyUpdRecordsetTestTable
    Group by MyUpdRecordsetTestTable.fieldForGrouping;
    info("Time needed: " + int2str(WinAPI::getTickCount()-tc));

    tc = WinAPI::getTickCount();
    update_recordSet MyUpdRecordsetTestTable setting theSum = MyUpdRecordsetTestTableTmp.theSum
    join MyUpdRecordsetTestTableTmp
```

```
    where MyUpdRecordsetTestTable.fieldForGrouping == MyUpdRecordsetTestTableTmp.
fieldForGrouping;
    info("Time needed: " + int2str(WinAPI::getTickCount()-tc));
}
```

When this code ran on demo data, the execution time of the operation was as follows:

- ***insert_recordset* statement** 1,685 milliseconds
- ***update_recordset* statement** 3,697 milliseconds

Restartable jobs and optimistic concurrency

In multiple scenarios in Microsoft Dynamics AX, the execution of some application logic involves manipulating multiple rows from the same table. Some scenarios require that all rows be manipulated within the scope of a single transaction. In such a scenario, if something fails and the transaction is cancelled, all modifications are rolled back, and the job can be restarted manually or automatically. Other scenarios commit the changes on a record-by-record basis. In the case of failure in these scenarios, only the changes to the current record are rolled back, and all previously manipulated records are committed. When a job is restarted in this scenario, it starts where it left off by skipping the records that have already changed.

An example of the first scenario is shown in the following code, in which all *update* queries to records in the CustTable table are wrapped into the scope of a single transaction:

```
static void UpdateCreditMax(Args _args)
{
    CustTable   custTable;

    ttsBegin;
    while select forupdate custTable where custTable.CreditMax == 0
    {
        if (custTable.balanceMST() < 10000)
        {
            custTable.CreditMax = 50000;
            custTable.update();
        }
    }
    ttsCommit;
}
```

An example of the second scenario, executing the same logic, is shown in the following code, in which the transaction scope is handled on a record-by-record basis. You must reselect each individual CustTable record inside the transaction for the Microsoft Dynamics AX run time to allow the record to be updated:

```
static void UpdateCreditMax(Args _args)
{
    CustTable   custTable;
    CustTable   updateableCustTable;
```

```
    while select custTable where custTable.CreditMax == 0
    {
        if (custTable.balanceMST() < 10000)
        {
            ttsBegin;
            select forupdate updateableCustTable
                where updateableCustTable.AccountNum == custTable.AccountNum;

            updateableCustTable.CreditMax = 50000;
            updateableCustTable.update();
            ttsCommit;
        }
    }
}
```

In a scenario in which 100 CustTable records qualify for the update, the first example would involve 1 *select* statement and 100 *update* statements being passed to the database, and the second example would involve 1 large *select* query and 100 additional *select* queries, plus the 100 *update* statements. The code in the first scenario would execute faster than the code in the second, but the first scenario would also hold the locks on the updated CustTable records longer because they wouldn't be committed on a record-by-record basis. The second example demonstrates superior concurrency over the first example because locks are held for a shorter time.

With the optimistic concurrency model in Microsoft Dynamics AX, you can take advantage of the benefits offered by both of the preceding examples. You can select records outside a transaction scope and update records inside a transaction scope—but only if the records are selected optimistically. In the following example, the *optimisticlock* keyword is applied to the *select* statement while maintaining a per-record transaction scope. Because the records are selected with the *optimisticlock* keyword, it isn't necessary to reselect each record individually within the transaction scope.

```
static void UpdateCreditMax(Args _args)
{
    CustTable   custTable;

    while select optimisticlock custTable where custTable.CreditMax == 0
    {
        if (custTable.balanceMST() < 10000)
        {
            ttsBegin;
            custTable.CreditMax = 50000;
            custTable.update();
            ttsCommit;
        }
    }
}
```

This approach provides the same number of statements passed to the database as in the first example, but with the improved concurrency from the second example because records are committed individually. The code in this example still doesn't perform as fast as the code in the first example because it has the extra burden of per-record transaction management. You could optimize

the example even further by committing records on a scale somewhere between all records and the single record, without decreasing the concurrency considerably. However, the appropriate choice for commit frequency always depends on the circumstances of the job.

Tip You can use the *forupdate* keyword when selecting records outside the transaction if the table has been enabled for optimistic concurrency at the table level. The best practice, however, is to use the *optimisticlock* keyword explicitly because the scenario won't fail if the table-level setting is changed. Using the *optimisticlock* keyword also improves the readability of the X++ code because the explicit intention of the developer is stated in the code.

Caching

The Microsoft Dynamics AX run time supports both single-record and set-based record caching. You can enable set-based caching in metadata by switching a property on a table definition or writing explicit X++ code that instantiates a cache. Regardless of how you set up caching, you don't need to know which caching method is used because the run time handles the cache transparently. To optimize the use of the cache, however, you must understand how each caching mechanism works.

Microsoft Dynamics AX 2012 introduces some important new features for caching. For example, record-based caching works not only for a single record but for joins as well. This mechanism is described in the "Record caching" section, which follows next. Also, even if range operations are used in a query, caching is supported so long as the query contains a unique key lookup.

The Microsoft Dynamics AX 2012 software development kit (SDK) contains a good description of the individual caching options and how they are set up. See the topic "Record Caching" at *http://msdn.microsoft.com/en-us/library/bb278240.aspx*.

This section focuses on how the caches are implemented in the Microsoft Dynamics AX run time and what to expect when using specific caching mechanisms.

Record caching

You can set up three types of record caching on a table by setting the *CacheLookup* property on the table definition: *Found, FoundAndEmpty,* and *NotInTTS.* An additional value (besides *None*) is *EntireTable*—a set-based caching option. These settings were introduced briefly in the section "Caching and indexing," earlier in this chapter, and are discussed in greater detail in this section.

The three types of record caching are fundamentally the same. The differences are found in what is cached and when cached values are flushed. For example, the *Found* and *FoundAndEmpty* caches are preserved across transaction boundaries, but a table that uses the *NotInTTS* cache doesn't use the cache when the cache is first accessed inside a transaction scope. Instead, the cache is used in consecutive *select* statements unless a *forupdate* keyword is applied to the *select* statement. (The *forupdate* keyword forces the run time to look up the record in the database because the previously cached record wasn't selected with the *forupdate* keyword applied.)

The following X++ code example illustrates when the cache is used inside a transaction scope when a table uses the *NotInTTS* caching mechanism. The *AccountNum* field is the primary key. The code comments indicate when the cache is used. In the example, the first two *select* statements after the *ttsbegin* command don't use the cache. The first statement doesn't use the cache because it's the first statement inside the transaction scope; the second doesn't use the cache because the *forupdate* keyword is applied to the statement.

```
static void NotInTTSCache(Args _args)
{
    CustTable custTable;

    select custTable                          // Look up in cache. If record
        where custTable.AccountNum == '1101'; // does not exist, look up
                                              // in database.

    ttsBegin;                                 // Start transaction.

    select custTable                          // Cache is invalid. Look up in
        where custTable.AccountNum == '1101'; // database and place in cache.

    select forupdate custTable                // Look up in database because
        where custTable.AccountNum == '1101'; // forupdate keyword is applied.

    select custTable                          // Cache will be used.
        where custTable.AccountNum == '1101'; // No lookup in database.

    select forupdate custTable                // Cache will be used because
        where custTable.AccountNum == '1101'; // forupdate keyword was used
                                              // previously.

    ttsCommit;                                // End transaction.

    select custTable                          // Cache will be used.
        where custTable.AccountNum == '1101';
}
```

If the table in the preceding example had been set up with *Found* or *FoundAndEmpty* caching, the cache would have been used when the first *select* statement was executed inside the transaction, but not when the first *select forupdate* statement was executed.

Note By default, all Microsoft Dynamics AX system tables are set up using a *Found* cache. This cannot be changed.

For all three caching mechanisms, the cache is used only if the *select* statement contains *equal-to* (==) predicates in the *where* clause that exactly match all of the fields in the primary index of the table or any one of the unique indexes that is defined for the table. Therefore, the *PrimaryIndex* property on the table must be set correctly on one of the unique indexes that is used when accessing the cache from application logic. For all other unique indexes, without any additional settings in metadata, the kernel automatically uses the cache if it is already present.

The following X++ code examples show when the Microsoft Dynamics AX run time will try to use the cache. The cache is used only in the first *select* statement; the remaining three statements don't match the fields in the primary index, so instead, the statements perform lookups in the database.

```
static void UtilizeCache(Args _args)
{
    CustTable custTable;

    select custTable                                    // Will use cache because only
        where custTable.AccountNum == '1101';           // the primary key is used as
                                                        // predicate.

    select custTable;                                   // Cannot use cache because no
                                                        // "where" clause exists.

    select custTable                                    // Cannot use cache because
        where custTable.AccountNum > '1101';            // equal-to (==) is not used.

    select custTable                                    // Will use cache even if
        where custTable.AccountNum == '1101'            // where clause contains more
        &&    custTable.CustGroup  == '20';             // predicates than the primary
                                                        // key. This assumes that the record
                                                        // have been successfully cached
                                                        // before. Please see the next sample.
}
```

Note The *RecId* index, which is always unique on a table, can be set as the *PrimaryIndex* in the table's properties. You can therefore set up caching by using the *RecId* field.

The following example illustrates how the improved caching mechanics in Microsoft Dynamics AX 2012 work when the *where* clause of the query contains more than just the unique index key columns:

```
static void whenRecordDoesGetCached(Args _args)
{
    CustTable custTable,custTable2;

    // Using Contoso demo data
    // The following select statement will not cache using the found cache because the lookup
    // will not return a record.
    // It would cache the record if the cache setting was FoundAndEmpty.

    select custTable
        where custTable.AccountNum == '1101'
        &&    custTable.CustGroup  == '20';

    // Following query will cache the record.

    select custTable
        where custTable.AccountNum == '1101';

    // Following will be cached too as the lookup will return a record.
```

```
    select custTable2
        where custTable2.AccountNum == '1101'
        &&    custTable2.CustGroup  == '10';

        // If you rerun the job, everything will come from the cache.
}
```

The following X++ code example shows how unique index caching works in the Microsoft Dynamics AX run time. The InventDim table in the base application has *InventDimId* as the primary key and a combination of keys (*inventBatchId*, *wmsLocationId*, *wmsPalletId*, *inventSerialId*, *inventLocationId*, *configId*, *inventSizeId*, *inventColorId*, and *inventSiteId*) as the unique index on the table.

Note This sample is based on Microsoft Dynamics AX 2012. The index has been changed for Microsoft Dynamics AX 2012 R2.

```
static void UtilizeUniqueIndexCache(Args _args)
{
    InventDim InventDim;
    InventDim inventdim2;

    select firstonly * from inventdim2;

    // Will use the cache because only the primary key is used as predicate

    select inventDim
    where inventDim.InventDimId == inventdim2.InventDimId;
    info(enum2str(inventDim.wasCached()));

    // Will use the cache because the column list in the where clause matches that of a unique
    // index
    // for the InventDim table and the key values point to same record as the primary key fetch

    select inventDim
     where inventDim.inventBatchId == inventDim2.inventBatchId
     && inventDim.wmsLocationId    == inventDim2.wmsLocationId
     && inventDim.wmsPalletId      == inventDim2.wmsPalletId
     && inventDim.inventSerialId   == inventDim2.inventSerialId
     && inventDim.inventLocationId == inventDim2.inventLocationId
     && inventDim.ConfigId         == inventDim2.ConfigId
     && inventDim.inventSizeId     == inventDim2.inventSizeId
     && inventDim.inventColorId    == inventDim2.inventColorId
     && inventDim.inventSiteId     == inventDim2.inventSiteId;
    info(enum2str(inventDim.wasCached()));

    // Cannot use cache because the where clause does not match the unique key list or primary
    // key.

    select firstonly inventDim
    where inventDim.inventLocationId== inventDim2.inventLocationId
     && inventDim.ConfigId          == inventDim2.ConfigId
     && inventDim.inventSiteId      == inventDim2.inventSiteId;
    info(enum2str(inventDim.wasCached()));
}
```

The Microsoft Dynamics AX run time ensures that all fields in a record are selected before they are cached. Therefore, if the run time can't find the record in the cache, it always modifies the field list to include all fields in the table before submitting the *SELECT* statement to the database. The following X++ code illustrates this behavior:

```
static void expandingFieldList(Args _args)
{
    CustTable custTable;

    select CreditRating  // The field list will be expanded to all fields.
        from custTable
        where custTable.AccountNum == '1101';
}
```

Expanding the field list ensures that the record fetched from the database contains values for all fields before the record is inserted into the cache. Even though the performance when fetching all fields is inferior compared to the performance when fetching a few fields, this approach is acceptable because in subsequent use of the cache, the performance gain outweighs the initial loss of populating it.

Tip You can avoid using the cache by calling the *disableCache* method on the record buffer with a Boolean parameter of *true*. This method forces the run time to look up the record in the database, and it also prevents the run time from expanding the field list.

The Microsoft Dynamics AX run time creates and uses caches on both the client tier and the server tier. The client-side cache is local to the Microsoft Dynamics AX client, and the server-side cache is shared among all connections to the server, including connections coming from Microsoft Dynamics AX Windows clients, web clients, the .NET Business Connector, and any other connection.

The cache that is used depends on the tier that the lookup is made from. If the lookup is executed on the server tier, the server-side cache is used. If the lookup is executed on the client tier, the client first looks in the client-side cache. If no record is found in the client-side cache, it executes a lookup in the server-side cache. If no record is found, a lookup is made in the database. When the database returns the record to the server and sends it on to the client, the record is inserted into both the server-side cache and the client-side cache.

If caching was set in Microsoft Dynamics AX 2009, the client stored up to 100 records per table, and the AOS stored up to 2,000 records per table. In Microsoft Dynamics AX 2012, you can configure the cache by using the Server Configuration form (System Administration > Setup > Server Configuration). For more information, see the section "Performance configuration options" later in this chapter.

Scenarios that perform multiple lookups on the same records and expect to find results in the cache can suffer performance degradation if the cache is continuously full—not only because records won't be found in the cache because they were removed based on the aging scheme, forcing a lookup in the database, but also because of the constant scanning of the tree to remove the oldest records. The following X++ code shows an example in which all SalesTable records are iterated through twice: each loop looks up the associated CustTable record. If this X++ code were executed on

the server and the number of lookups for CustTable records was more than 2,000 (assuming that the cache was set to 2,000 records on the server), the oldest records would be removed from the cache and the cache would no longer contain all CustTable records when the first loop ended. When the code iterates through the SalesTable records again, the records might not be in the cache, and the run time would go to the database to look up the CustTable records. The scenario, therefore, would perform much better with fewer than 2,000 records in the database.

```
static void AgingScheme(Args _args)
{
    SalesTable salesTable;
    CustTable  custTable;

    while select salesTable order by CustAccount
    {
        select custTable         // Fill up cache.
            where custTable.AccountNum == salesTable.CustAccount;

        // More code here.
    }

    while select salesTable order by CustAccount
    {
        select custTable         // Record might not be in cache.
            where custTable.AccountNum == salesTable.CustAccount;

        // More code here.
    }

}
```

Important Test performance improvements of record caching only on a database where the database size and data distribution resemble the production environment. (The arguments have been presented in the previous example.)

Before the Microsoft Dynamics AX run time searches for, inserts, updates, or deletes records in the cache, it places a mutually exclusive lock that isn't released until the operation is complete. This lock means that two processes running on the same server can't perform insert, update, or delete operations in the cache at the same time. Only one process can hold the lock at any given time, and the remaining processes are blocked. Blocking occurs only when the run time accesses the server-side cache. So although the caching possibilities supported by the run time are useful, you should use them only when appropriate. If you can reuse a record buffer that is already fetched, you should do so. The following X++ code shows the same record fetched multiple times. The subsequent fetch operations use the cache, even though it could have used the first fetched record buffer.

```
static void ReuseRecordBuffer(Args _args)
{
    CustTable    custTable;
    CurrencyCode myCustCurrency;
    CustGroupId  myCustGroupId;
```

```
    PaymTermId   myCustPaymTermId;

    // Bad coding pattern

    myCustGroupId = custTable::find('1101').CustGroup;
    myCustPaymTermId = custTable::find('1101').PaymTermId;
    myCustCurrency = custTable::find('1101').Currency;

    // The cache will be used for these lookups, but it is much more
    // efficient to reuse the buffer, because even cache lookups are not "free."
    // Good coding pattern:

    custTable        = CustTable::find('1101');
    myCustGroupId    = custTable.CustGroup;
    myCustPaymTermId = custTable.PaymTermId;
    myCustCurrency   = custTable.Currency;
}
```

The unique index join cache

The unique index join cache is new to Microsoft Dynamics AX 2012 and allows caching of subtype and supertype tables, one-to-one relation joins with a unique lookup, or a combination of both. A key constraint with this type of cache is that you can look up only one record through a unique index and you can join only over unique columns.

The following example illustrates all three possible variations:

```
public static void main(Args args)
{
    SalesTable      header;
    SalesLine       line;
    DirPartyTable   party;
    CustTable       customer;
    int             i;

        // subtype, supertype table caching

        for (i=0 ; i<1000; i++)
            select party where party.RecId == 5637144829;

        // 1:1 join data caching

        for (i=0 ; i<1000; i++)
            select line
            join header
            where line.RecId == 5637144586
               && line.SalesId == header.SalesId;

        // Combination of subtype, supertype, and 1:1 join caching

        for (i=0 ; i<1000; i++)
            select customer
            join party
            where customer.AccountNum == '4000'
               && customer.Party == party.RecId;
}
```

The *EntireTable* cache

In addition to using the three caching methods described so far—*Found, FoundAndEmpty,* and *NotInTTS*—you can set a fourth caching option, *EntireTable*, on a table. *EntireTable* enables a set-based cache. It causes the AOS to mirror the table in the database by selecting all records in the table and inserting them into a temporary table when any record from the table is selected for the first time. The first process to read from the table can therefore experience a longer response time because the run time reads all records from the database. Subsequent *select* queries then read from the *EntireTable* cache instead of from the database.

A temporary table is usually local to the process that uses it, but the *EntireTable* cache is shared among all processes that access the same AOS. Each company (as defined by the *DataAreaId* field) has an *EntireTable* cache, so two processes requesting records from the same table but from different companies use different caches, and both could experience a longer response time to instantiate the entire-table cache.

The *EntireTable* cache is a server-side cache only. When the run time requests records from the client tier on a table that is *EntireTable* cached, the table behaves like a *Found* cached table. If a request for a record is made on the client tier that qualifies for searching the record cache, the client first searches the local *Found* cache. If the record isn't found, the client calls the AOS to search the *EntireTable* cache. When the run time returns the record to the client tier, it inserts the record into the client-side *Found* cache. The *EntireTable* cache on the server side uses a *Found* cache in addition when unique key lookups are made.

The *EntireTable* cache isn't used in the execution of a *select* statement that joins a table that is *EntireTable* cached to a table that isn't *EntireTable* cached. In this situation, the *select* statement is passed to the database. However, when *select* statements are made that access only a single table that is *EntireTable* cached, or when joining other tables that are *EntireTable* cached, the *EntireTable* cache is used.

The Microsoft Dynamics AX run time flushes the *EntireTable* cache when records are inserted, updated, or deleted in the table. The next process that selects records from the table suffers degraded performance because it must reread the entire table into the cache. In addition to flushing its own cache, the AOS that executes the insert, update, or delete also informs other AOS instances in the same installation that they must flush their caches of the same table. This prevents old and invalid data from being cached for too long. In addition to this flushing mechanism, the AOS flushes all *EntireTable* caches every 24 hours.

Because of the flushing that results when modifying records in a table that has been *EntireTable* cached, avoid setting up *EntireTable* caches on frequently updated tables. Rereading all records into the cache results in a performance loss, which could outweigh the performance gain achieved by caching records on the server tier and avoiding round-trips to the database tier. You can overwrite the *EntireTable* cache setting on a specific table at run time when you configure Microsoft Dynamics AX.

Even if the records in a table are fairly static, you might achieve better performance by not using an *EntireTable* cache if the table has a large number of records. Because an *EntireTable* cache uses temporary tables, it changes from an in-memory structure to a file-based structure when the table

uses more than 128 kilobytes (KB) of memory. This results in performance degradation during record searches. The database search engines have also evolved over time and are faster than the ones implemented in the Microsoft Dynamics AX run time. It might be faster to let the database search for the records than to set up and use an *EntireTable* cache, even though a database search involves round-trips to the database tier. In Microsoft Dynamics AX 2012, you can configure the amount of memory an entire table can consume before it changes to a file-based structure. To do so, go to System Administration > Setup > System > Server Configuration.

The *RecordViewCache* class

A *RecordViewCache* object is implemented as a linked list that allows only a sequential search for records. When you use the cache to store a large number of records, search performance is degraded because of this linked-list format. Therefore, you should not use it to cache more than 100 records. Weigh the use of the cache against the extra time spent fetching the records from the database, which uses a more optimal search algorithm. In particular, consider the time required when you search for only a subset of records; the Microsoft Dynamics AX run time must continuously match each record in the cache against the more granular *where* clause in the *select* statement because no indexing is available for the records in the cache.

You can use the *RecordViewCache* class to establish a set-based cache from X++ code. You initiate the cache by writing the following X++ code:

```
select nofetch custTrans where custTrans.accountNum == '1101';
recordViewCache = new RecordViewCache(custTrans);
```

The records to cache are described in the *select* statement, which must include the *nofetch* keyword to prevent the selection of the records from the database. The records are selected when the *RecordViewCache* object is instantiated with the record buffer passed as a parameter. Until the *RecordViewCache* object is destroyed, *select* statements will execute on the cache if they match the *where* clause defined when the cache was instantiated. The following X++ code shows how to instantiate and use the cache:

```
public static void main(Args _args)
{
    InventTrans     inventTrans;
    RecordViewCache recordViewCache;
    int countNone, countSold, countOrder;

    // Define records to cache.

    select nofetch inventTrans
       where inventTrans.ItemId == '1001';

    // Cache the records.

    recordViewCache = new RecordViewCache(InventTrans);

    // Use the cache.

    while select inventTrans
```

```
        index hint ItemIdx
        where inventTrans.ItemId == '1001' && inventTrans.StatusIssue == StatusIssue::OnOrder
    {
        countOrder++;

        //Additional code here

    }

    // This block of code needs to be executed only after the first while select statement and
    // before the second while select statement.

    // Additonal code here

    // Uses the cache again.

    while select  inventTrans
        index hint ItemIdx
        where inventTrans.ItemId == '1001' && inventTrans.StatusIssue == StatusIssue::Sold
    {
        countSold++;
        //Additional code here
    }
    info('OnOrder Vs Sold = '+int2str(countOrder) + ' : ' + int2str(countSold));
}
```

The cache can be instantiated only on the server tier. The *select* statement can contain only *equal-to* (==) predicates in the *where* clause and is accessible only by the process instantiating the cache object. If the table buffer used for instantiating the cache object is a temporary table or if it uses *EntireTable* caching, the *RecordViewCache* object isn't instantiated.

If the table that is cached in the *RecordViewCache* object is also cached on a per-record basis, the run time can use both caches. If a *select* statement is executed on a *Found* cached table and the *select* statement qualifies for lookup in the *Found* cache, the run time performs a lookup in this cache first. If no record is found and the *select* statement also qualifies for lookup in the *RecordViewCache* object, the run time uses the *RecordViewCache* object and updates the *Found* cache after retrieving the record.

Inserts, updates, and deletions of records that meet the cache criteria are reflected in the cache at the same time that the data manipulation language (DML) statements are sent to the database. Records in the cache are always inserted at the end of the linked list. A hazard associated with this behavior is that an infinite loop can occur when application logic iterates through the records in the cache and at the same time inserts new records that meet the cache criteria.

Changes to records in a *RecordViewCache* object can't be rolled back. If one or more *RecordViewCache* objects exist, if the *ttsabort* operation executes, or if an error is thrown that results in a rollback of the database, the *RecordViewCache* objects still contain the same information. Therefore, any instantiated *RecordViewCache* object that is subject to modification by application logic should not have a lifetime longer than the transaction scope in which it is modified. The *RecordViewCache* object must be declared in a method that isn't executed until after the transaction has begun. In the event of a rollback, the object and the cache are both destroyed.

SysGlobalObjectCache and *SysGlobalCache*

Microsoft Dynamics AX 2012 provides two mechanisms that you can use to cache global variables to improve performance: *SysGlobalObjectCache (SGOC)* and *SysGlobalCache*. *SGOC* is new for Microsoft Dynamics AX 2012 and is an important performance feature.

SGOC is a global cache that is located on the AOS, and not just a session-based cache. You can use this cache to reduce round-trips to the database or to store intermediate calculation results. The data that is stored from one user connection is available for all users. For more information about the SGOC, see the entry "Using SysGlobalObjectCache (SGOC) and understanding its performance implications" on the Microsoft Dynamics AX Performance Team blog (*http://blogs.msdn.com/b/axperf/archive/2011/12/29/using-sysglobalobjectcache-sgoc-and-understanding-it-s-performance-implications.aspx*).

SysGlobalCache uses a map to save information that is purely session-based. However, there are certain client/server considerations if you use this form of caching. If you use *SysGlobalCache* by means of the *ClassFactory* class, global variables can exist either on the client or on the server. If you use *SysGlobalCache* directly, it runs on the tier from which it is called. If you use *SysGlobalCache* by means of the *Info* class or the *Application* class, it resides on both tiers, causing a performance penalty because of increased round-trips between the client and server. For more information, see "Using Global Variables" at *http://msdn.microsoft.com/en-us/library/aa891830.aspx*.

Field lists

Most X++ *select* statements in Microsoft Dynamics AX retrieve all fields for a record, even though only a few of the fields are actually used. The main reason for this coding style is that the Microsoft Dynamics AX run time doesn't report compile-time or run-time errors if a field on a record buffer is accessed and hasn't been retrieved from the database. Because of the normalization of the Microsoft Dynamics AX 2012 data model and the introduction of table hierarchies, limiting field lists in queries is even more important than it was in Microsoft Dynamics AX 2009, particularly for polymorphic tables. With ad hoc mode, you can limit the field list in a query. If you use ad hoc mode, the query is limited to only the table (or tables) that are referenced in the query. Other tables in the hierarchy are excluded. This produces an important performance benefit by reducing the number of joins between tables in subtype and supertype hierarchies.

Note The base type table is always joined, regardless of which fields are selected.

The following example illustrates the effects of querying both without and with ad hoc mode:

```
static void AdHocModeSample(Args _args)
{
    DirPartyTable dirPartyTable;
    CustTable     custTable;
    select dirPartyTable join custTable where dirPartyTable.RecId==custTable.Party;

    /*Would result in the following query to the database:
```

```
    SELECT T1.NAME,
   T1.LANGUAGEID,

--<...Fields removed for better readability. Basically, all fields from all tables would be
fetched...>

   T9.MEMO FROM DIRPARTYTABLE T1 LEFT OUTER JOIN DIRPERSON T2 ON (T1.RECID=T2.RECID) LEFT
OUTER JOIN DIRORGANIZATIONBASE T3 ON (T1.RECID=T3.RECID) LEFT OUTER JOIN DIRORGANIZATION T4 ON
(T3.RECID=T4.RECID) LEFT OUTER JOIN OMINTERNALORGANIZATION T5 ON (T3.RECID=T5.RECID) LEFT OUTER
JOIN OMTEAM T6 ON (T5.RECID=T6.RECID) LEFT OUTER JOIN OMOPERATINGUNIT T7 ON (T5.RECID=T7.RECID)
LEFT OUTER JOIN COMPANYINFO T8 ON (T5.RECID=T8.RECID) CROSS JOIN CUSTTABLE T9 WHERE
((T9.DATAAREAID='ceu') AND (T1.RECID=T9.PARTY))

    Limiting the field list will force the Microsoft Dynamics AX 2012 AOS to query only for the
actual table.
    The following query:*/

    select RecId from dirPartyTable exists join custTable where dirPartyTable.RecId==custTable.
Party;
    /*
Results only in the following query to SQL Server

    SELECT T1.RECID,         T1.INSTANCERELATIONTYPE FROM DIRPARTYTABLE T1 WHERE EXISTS (SELECT
'x' FROM CUSTTABLE T2 WHERE ((T2.DATAAREAID='ceu') AND (T1.RECID=T2.PARTY)))
    */
}
```

There are additional ways to limit the field list and number of joins in queries through the user interface. These are described in more detail at the end of this section.

The following X++ code, which selects only the *AccountNum* field from the CustTable table but evaluates the value of the *CreditRating* field and sets the *CreditMax* field, won't fail because the run time doesn't detect that the fields haven't been selected:

```
static void UpdateCreditMax(Args _args)
{
    CustTable custTable;

    ttsBegin;
    while select forupdate AccountNum from custTable
    {
        if (custTable.CreditRating == '')
        {
            custTable.CreditMax = custTable.CreditMax + 1000;
            custTable.update();
        }
    }
    ttsCommit;
}
```

This code adds *1,000* to the value of the *CreditMax* field in CustTable records for which the *CreditRating* field is empty. However, adding the *CreditRating* and *CreditMax* fields to the field list of

the *select* statement might not solve the problem: the application logic could still update other fields incorrectly because the *update* method on the table could be evaluating and setting other fields in the same record.

Important You could examine the *update* method for other fields accessed in the method and then select these fields also, but new problems would soon surface. For example, if you customize the *update* method to include application logic that uses additional fields, you might not be aware that the X++ code in the preceding example also needs to be customized.

Limiting the field list when selecting records results is a performance gain because less data is retrieved from the database and sent to the AOS. The gain is even greater if you can retrieve the fields by using indexes without a lookup of the values in the table or by limiting the field list to reduce the number of joins in hierarchy tables. You can implement this performance improvement and write *select* statements safely when you use the retrieved data within a controlled scope, such as a single method. The record buffer must be declared locally and not passed to other methods as a parameter. Any developer customizing the X++ code can easily see that only a few fields are selected and act accordingly.

To truly benefit from a limited field list, be aware that the Microsoft Dynamics AX run time sometimes automatically adds extra fields to the field list before passing a statement to the database. One example was explained earlier in this chapter in the "Caching" section. In that example, the run time expanded the field list to include all fields if the *select* statement qualifies for storing the retrieved record in the cache.

In the following X++ code, you can see how the Microsoft Dynamics AX run time adds additional fields. The code calculates the total balance for all customers in customer group *20* and converts the balance into the company's unit of currency. The *amountCur2MST* method converts the value in the currency specified in the *CurrencyCode* field to the company currency.

```
static void BalanceMST(Args _args)
{
    CustTable   custTable;
    CustTrans   custTrans;
    AmountMST   balanceAmountMST = 0;

    while select custTable
            where custTable.CustGroup == '20'
          join custTrans
            where custTrans.AccountNum == custTable.AccountNum
    {
        balanceAmountMST += Currency::amountCur2MST(custTrans.AmountCur,
                                                    custTrans.CurrencyCode);
    }
}
```

When the *select* statement is passed to the database, it retrieves all fields in the CustTable and CustTrans tables, even though only the *AmountCur* and *CurrencyCode* fields on the CustTrans table are used. The result is the retrieval of more than 100 fields from the database.

You can optimize the field list by selecting the *AmountCur* and *CurrencyCode* fields from the CustTrans table and, for example, only the *AccountNum* field from the CustTable table, as shown in the following code:

```
static void BalanceMST(Args _args)
{
    CustTable   custTable;
    CustTrans   custTrans;
    AmountMST   balanceAmountMST = 0;

    while select AccountNum from custTable
            where custTable.CustGroup == '20'
          join AmountCur, CurrencyCode from custTrans
            where custTrans.AccountNum == custTable.AccountNum
    {
        balanceAmountMST += Currency::amountCur2MST(custTrans.AmountCur,
                                                    custTrans.CurrencyCode);
    }
}
```

As explained earlier, the application run time expands the field list from the three fields shown in the preceding X++ code example to five fields because it adds the fields that are used when updating the records. These fields are added even though neither the *forupdate* keyword nor any of the specific concurrency model keywords are applied to the statement. The statement passed to the database starts as shown in the following example, in which the RECID column is added for both tables:

```
SELECT A.ACCOUNTNUM,A.RECID,B.AMOUNTCUR,B.CURRENCYCODE,B.RECID
FROM CUSTTABLE A,CUSTTRANS B
```

To prevent the retrieval of any fields from the CustTable table, you can rewrite the *select* statement to use the *exists join* operator, as shown here:

```
static void BalanceMST(Args _args)
{
    CustTable   custTable;
    CustTrans   custTrans;
    AmountMST   balanceAmountMST = 0;

    while select AmountCur, CurrencyCode from custTrans
        exists join custTable
            where custTable.CustGroup  == '20' &&
                  custTable.AccountNum == custTrans.AccountNum
    {
        balanceAmountMST += Currency::amountCur2MST(custTrans.AmountCur,
                                                    custTrans.CurrencyCode);
    }
}
```

This code retrieves only three fields (*AmountCur*, *CurrencyCode*, and *RecId*) from the CustTrans table and none from the CustTable table.

In some situations, however, it might not be possible to rewrite the statement to use *exists join*. In such cases, including only *TableId* as a field in the field list prevents the retrieval of any fields from the table. To do this, you modify the original example as follows to include the *TableId* field:

```
static void BalanceMST(Args _args)
{
    CustTable   custTable;
    CustTrans   custTrans;
    AmountMST   balanceAmountMST = 0;

    while select TableId from custTable
            where custTable.CustGroup == '20'
          join AmountCur, CurrencyCode from custTrans
            where custTrans.AccountNum == custTable.AccountNum
    {
        balanceAmountMST += Currency::amountCur2MST(custTrans.AmountCur,
                                                    custTrans.CurrencyCode);
    }
}
```

This code causes the Microsoft Dynamics AX run time to pass a *select* statement to the database with the following field list:

```
SELECT B.AMOUNTCUR,B.CURRENCYCODE,B.RECID
FROM CUSTTABLE A,CUSTTRANS B
```

If you rewrite the *select* statement to use *exists join* or include only *TableId* as a field, the *select* statement sent to the database retrieves just three fields instead of more than 100. As you can see, you can substantially improve your application's performance just by rewriting queries to retrieve only the necessary fields.

Tip You can use the Best Practice Parameters dialog box to have Microsoft Dynamics AX analyze the use of *select* statements in X++ code and recommend whether to implement field lists based on the number of fields that are accessed in the method. To enable this check, in the AOT or the Development Workspace, on the Tools menu, click Options > Development > Best Practices. In the Best Practice Parameters dialog box, make sure that AOS Performance Check is selected and that Warning Level is set to Errors and Warnings.

To use ad hoc mode on forms, navigate to *Data Sources* node for the form you want in the AOT, and then select the appropriate data source and set the *OnlyFetchActive* property to *Yes*, as shown in Figure 13-6. This setting limits the number of fields fetched to only those fields that are used by controls on the form and improves the form's response time. Additionally, if the data source is a polymorphic table, only the tables that are necessary to return these fields are joined—instead of all tables within the hierarchy.

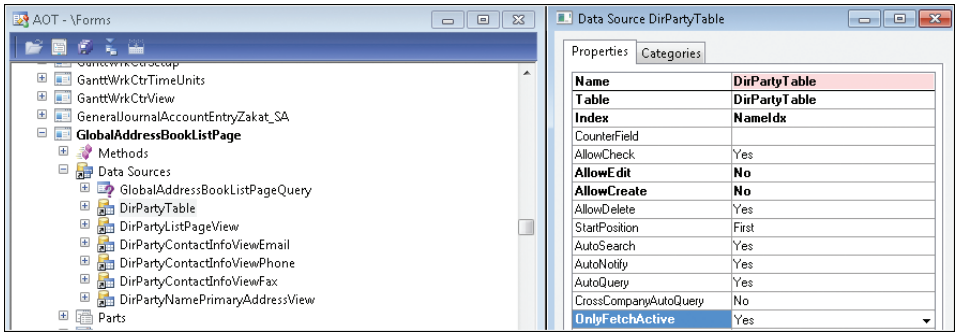


FIGURE 13-6 Use of *OnlyFetchActive* on a list page.

To see the effect of ad hoc mode, do the following test: create a list page containing the DirPartyTable table as the data source and add only three fields to the list page grid; for example, *Name*, *NameAlias*, and *PartyNumber*. Setting *OnlyFetchActive* to *No* results in the following query, which contains all fields in all tables and joins to all tables in the hierarchy:

```
SELECT T1.DEL_GENERATIONALSUFFIX,T1.NAME, T1.NAMEALIAS,T1.PARTYNUMBER,
/* Field list shortened for better readability. All fields of all tables would be fetched. */
T8.RECID, FROM DIRPARTYTABLE T1 LEFT OUTER JOIN DIRPERSON T2 ON (T1.RECID=T2.RECID) LEFT
OUTER JOIN DIRORGANIZATIONBASE T3 ON (T1.RECID=T3.RECID) LEFT OUTER JOIN DIRORGANIZATION T4 ON
(T3.RECID=T4.RECID) LEFT OUTER JOIN OMINTERNALORGANIZATION T5 ON (T3.RECID=T5.RECID) LEFT OUTER
JOIN OMTEAM T6 ON (T5.RECID=T6.RECID) LEFT OUTER JOIN OMOPERATINGUNIT T7 ON (T5.RECID=T7.RECID)
LEFT OUTER JOIN COMPANYINFO T8 ON (T5.RECID=T8.RECID)ORDER BY T1.PARTYNUMBER
```

Setting *OnlyFetchActive* to *Yes* results in a much smaller and more efficient query:

```
SELECT T1.NAME,T1.NAMEALIAS, T1.PARTYNUMBER, T1.RECID,T1.RECVERSION, T1.INSTANCERELATIONTYPE
FROM DIRPARTYTABLE T1        ORDER BY T1.PARTYNUMBER
```

For polymorphic tables in datasets for Enterprise Portal web controls, ensure that you also set *OnlyFetchActive* to *Yes* on the data source of the dataset to improve performance.

To use ad hoc mode on queries that are modeled in the AOT, do the following:

1. Navigate to the query you want, and then expand the *Data Sources* node and the appropriate data source.
2. Click the *Fields* node, and then set the *Dynamic* property to *No* (see Figure 13-7).

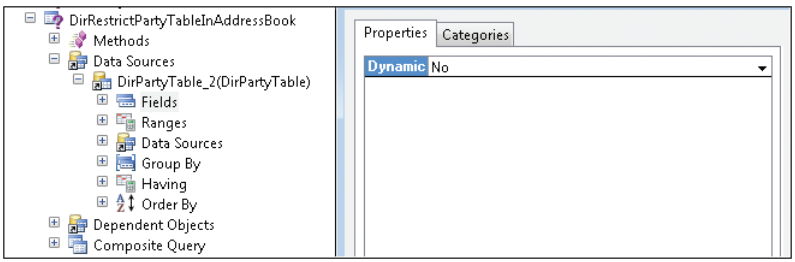


FIGURE 13-7 Use ad hoc mode on modeled queries.

3. Reduce the fields to only the ones that are necessary.

For an example of a query with a restricted field list, see the *DirRestrictPartyTableInAddressBook* query in the base application.

Field justification

Microsoft Dynamics AX supports left-and right-justification of extended data types. With Microsoft Dynamics AX 2012, nearly all extended data types are left justified to reduce the impact of space consumption because of double- and triple-byte storage as a result of Unicode enablement. Left justifying also helps performance by increasing the speed of access through indexes.

When sorting is critical, you can use right justification. However, you should use this technique sparingly.

Performance configuration options

This section provides an overview of the most important configuration options that can improve the performance of your Microsoft Dynamics AX 2012 installation.

SQL Administration form

The SQL Administration form (Figure 13-8) offers a set of SQL Server features that were not supported in previous versions of Microsoft Dynamics AX. For example, you can compress a table or apply a fill factor individually. The SQL Administration form is located under System Administration > Periodic > Database > SQL Administration.

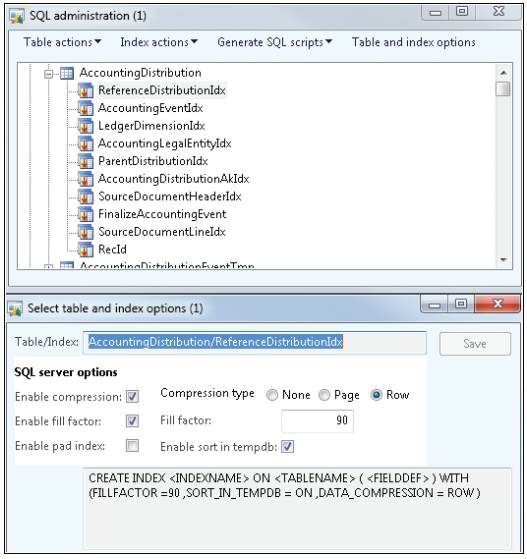


FIGURE 13-8 The SQL Administration form.

Server Configuration form

Several important performance options are located on the Server Configuration form. You can use this form to specify settings for performance optimization, batch operations, and caching. The Server Configuration form is located under System Administration > Setup > System > Server Configuration.

Some of the most important performance optimization options are as follows:

- **Maximum number of tables in join** Limits the number of tables you can have in a join. Too many joins can have a negative impact on performance, especially if the fields that are joined are not indexed well.
- **Client record cache factor** Determines how many records the client caches. For example, if the server-side cache setting for a table in the Main table group is set to 2,000, and you set this setting to *20*, then the client will cache 100 records (2,000/20).
- **Timeout for user modified queries** Specifies the timeout, in seconds, for queries when a user adds conditions by using the SysQueryForm form. A setting of *0* means that here is no timeout. If a query times out, a message is shown.

You can specify whether a server is a batch server and how many threads the server can use to process batch jobs. A good formula to determine how many batch threads a server can use is to multiply the number of cores by 2. The number of threads that a server can use depends on the processes that are running on the server. For some processes, the server can use more than two threads for each core. However, you need to test this on a case-by-case basis.

You can also define the number of records that are stored in a cache and other cache settings (Figure 13-9), such as the size of an *EntireTable* cache (in kilobytes), the maximum number of objects that the *SGOC* can hold, and the number of records that can be cached for each table group. Each server can have its own cache settings.

AOS configuration

The Microsoft Dynamics AX 2012 Server Configuration tool contains settings that you can use to improve the performance of the AOS. To access the tool, on the Start menu, click Administrative Tools > Microsoft Dynamics AX 2012 Server Configuration. The following options are some of the most important:

- **Application Object Server tab** Generally, the settings Enable Breakpoints To Debug X++ Code Running On This Server and Enable Global Breakpoints should be turned off in production systems. Enable The Hot-Swapping Of Assemblies For Each Development Session should also be turned off in production systems. All three of these options might cause a performance penalty if enabled.
- **Database Tuning tab** Depending on the business processes you run, increasing the value of the Statement Cache setting can improve or degrade performance. This setting determines how many statements the AOS caches. (Only the statements and not the result sets are cached.) You should not change the default value without thorough testing. Also, you should

avoid changing the Maximum Buffer Size setting because the larger the maximum buffer size, the more memory that must be allocated for each buffer, which takes slightly more time.

- **Performance tab** If you have multiple AOS instances on one server, use this tab to define an affinity to avoid resource contention between the AOS instances. Note that the AOS in Microsoft Dynamics AX 2012 can scale more than eight cores effectively.

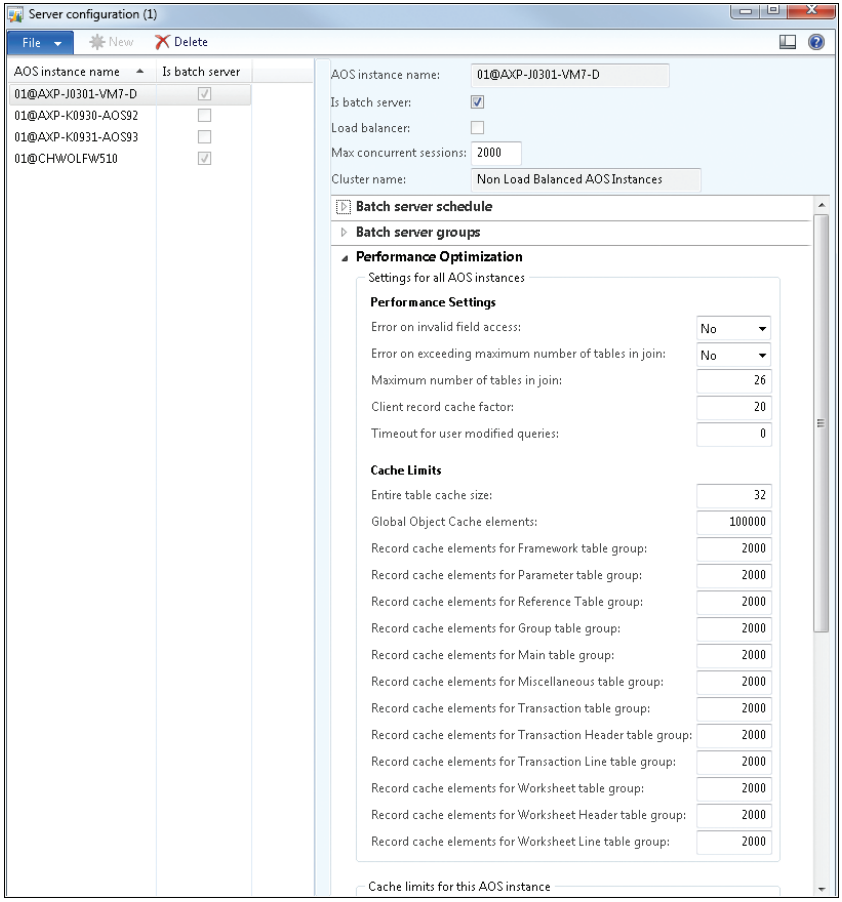


FIGURE 13-9 Caching options on the Server Configuration form.

Client configuration

On the AOS, you can use the Microsoft Dynamics AX Configuration tool to set options for Microsoft Dynamics AX clients. To access this tool, on the Start menu, click Administrative Tools > Microsoft Dynamics AX 2012 Configuration. On the Performance tab, under Cache Settings, if you select the Least Memory setting, the loading of certain dynamic-link libraries (DLLs) will be deferred until they are needed, to save memory. This setting slightly decreases performance but is very useful in Terminal Services scenarios to increase the number of users that a Terminal Server can host in parallel.

Client performance

You can use the Client Performance Options form to centrally disable a set of features that might affect performance. You can access the form under System Administration > Setup > System > Client Performance Options.

For a detailed description of the controls on this form, see the entry "Microsoft Dynamics AX 2012: Client Performance Options" on the Microsoft Dynamics AX Performance Team blog (*http://blogs.msdn.com/b/axperf/archive/2011/11/07/ax2012-client-performance-options.aspx*).

Number sequence caching

It is a best practice to review thoroughly all number sequences that are in use to determine whether they should be continuous. If possible, set them to be non-continuous. All number sequences that are not continuous should have caching enabled.

Under Organization Administration > Common > Number Sequences, double-click the number sequence you want, and then on the Performance FastTab, set a preallocation depending on the frequency with which the number sequence is used.

Extensive logging

Extensive database logging and other logging mechanisms, such as the sales and marketing transaction log (Sales and Marketing > Setup > Sales and Marketing Parameters), add overhead to the database load and should be reduced to the absolute minimum necessary.

Master scheduling and inventory closing

Microsoft Dynamics AX 2012 has optimized performance of the master scheduling and inventory closing processes. Both processes should run at least with one helper thread. However, it is better to use multiple helper threads. For master scheduling, eight helper threads have been found to be optimum with the tested data.

Another option to improve the speed of master scheduling is to have a dedicated AOS and change the garbage collection pattern to client-based garbage collection. To do so, navigate to the installation directory of the appropriate AOS, and then open the Ax32Serv.exe.config file.

Locate the following XML node and set it to *false*:

```
<gcServer enabled="true" />
```

Coding patterns for performance

This section discusses coding patterns that you can use to help optimize performance.

Execute X++ code as CIL

You can improve performance by running X++ as common intermediate language (CIL). In general, if a service is called from outside Microsoft Dynamics AX 2012, it is executed in CIL. Batch jobs, services in the AxClient service group, and code that is traversed through the *RunAs* method are also executed in CIL. Two interfaces are available for this purpose in Classes\Global\runClassMethodIL and runTableMethodIL. The performance benefit from running X++ in CIL comes mainly from better .NET garbage collection. Depending on your process, the performance improvement can be between 0 and 30 percent. Therefore, you'll need to test to see whether performance improves by running your process in CIL.

Use parallel execution effectively

Microsoft Dynamics AX 2009 introduced ways to implement parallel processing easily through the batch framework. These options have been enhanced in Microsoft Dynamics AX 2012. Three common patterns can be applied for scheduling batch jobs that execute tasks in parallel: batch bundling, individual task modeling, and top picking. Each pattern has its own advantages and disadvantages, which are discussed in the following sections.

For more information about the batch framework, see Chapter 18, "The batch framework." For code samples and additional information about batch patterns and performance, see the entry "Batch Parallelism Microsoft Dynamics AX – Part I" on the Microsoft Dynamics AX Performance Team blog (*http://blogs.msdn.com/b/axperf/archive/2012/02/24/batch-parallelism-in-ax-part-i.aspx*). Links to additional entries in this series are provided in the following sections.

Batch bundling

With batch bundling, you create a static number of tasks and split the work among these tasks by grouping the work items into bundles. The workload distribution between each task should be as equal as possible. Each worker thread processes a bundle of work items before picking up the next bundle. This pattern works well if all of the tasks take roughly the same amount of time to process in each bundle. In an ideal situation, each worker thread is actively doing the same amount of work. But in scenarios where the workload is variable because of data composition or differences in server hardware, this approach is not the most efficient. In these scenarios, the last few threads might take longer to complete because they are processing larger bundles than the others.

You can find a code example illustrating batch bundling in the AOT at Classes\FormletterService-BatchTaskManager\createFormletterParmDataTasks().

Individual task modeling

With individual task modeling, parallel processing is achieved by creating a separate task for each work item so that there is a one-to-one mapping between the task and the work item. This eliminates the need for preallocation. Because each work item is independently handled by a worker thread, workload distribution is more consistent. This approach eliminates the problem of a number of large work items being bundled together and eventually increasing the response time for the batch.

This pattern is not necessarily suitable for processing a large number of work items because you will end up with a large number of batch tasks. The overhead on the batch framework to maintain a large number of tasks is high because the batch framework must check several conditions, dependencies, and constraints whenever a set of tasks is completed and a new set of tasks must be picked up for execution from the ready state.

You can find a code example that illustrates this pattern on the Microsoft Dynamics AX Performance Team blog (*http://blogs.msdn.com/b/axperf/archive/2012/02/25/batch-parallelism-in-ax-part-ii.aspx*).

Top picking

One issue with bundling is the uneven distribution of workload. You can address that by using individual task modeling, but that can produce high overhead on the batch framework. Top picking is another batching technique that can address the problem of uneven workload distribution. However, it causes the same problem as individual task modeling with a large number of work items.

.With top picking, a static number of tasks are created—just as in bundling—and preallocation is unnecessary—just as in individual task modeling. Because no preallocation is performed, the pattern does not rely on the batch framework to separate the work items, but you do need to maintain a staging table to track the progress of the work items. Maintaining the staging table has its own overhead, but that overhead is much lower than the overhead of the batch framework. After the staging table is populated, the worker threads start processing by fetching the next available item from the staging table and continue until no work items are left. This means that no worker threads are idle while other worker threads are overloaded. To implement top picking, you use the *PESSIMISTICLOCK* hint along with the *READPAST* hint. Used together, these hints enable worker threads to fetch the next available work item without being blocked.

You can find a code example that illustrates this pattern on the Microsoft Dynamics AX Performance Team blog (*http://blogs.msdn.com/b/axperf/archive/2012/02/28/batch-parallelism-in-ax-part-iii.aspx*).

The SysOperation framework

In Microsoft Dynamics AX 2012, programming concepts are available and first steps have been taken to replace the RunBase framework. By using its replacement, the SysOperation framework, you can run services in Microsoft Dynamics AX in various execution modes. The SysOperation framework has performance advantages, too. There is a clear separation of responsibilities between tiers, and execution happens solely on the server tier. These enhancements ensure a minimum number of round-trips.

Note Chapter 14 contains more information about the SysOperation framework and additional code sample that compares the SysOperation framework with the RunBase framework. If you are unfamiliar with the SysOperation framework, it is recommended that you read Chapter 14 before you read this section.

The SysOperation framework supports four execution modes:

- **Synchronous** You can run a service in synchronous mode on the server. The client waits until the process on the server is complete, and only then can the user continue working.
- **Asynchronous** You perform the necessary configurations to the data contract and then execute code on the server. However, the client remains responsive and the user can continue working. This mode also saves round-trips between the client and the server.
- **Reliable asynchronous** Running operations in this mode is equivalent to running them on the batch server, with the additional behavior that the jobs are deleted after they are completed (regardless of whether they are successful). The job history remains. This pattern facilitates building operations that use the batch server run time, but that do not rely on the batch server administration features.
- **Scheduled batch** You use this mode for scheduled batch jobs that run on a regular basis.

The following example illustrates how to calculate a set of prime numbers. A user enters the starting number (such as 1,000,000) and an ending number (such as 1,500,000). The service then calculates all prime numbers in that range. This example will be used to illustrate the differences in timing when running an execution in each mode. The sample consists of two classes (a service class and a data contract), a table to store the results, and a job and an enumerator to demonstrate the execution and execution modes.

Note Instead of using a job, you would typically use menu items to execute the operation. If you use a menu item, the SysOperation framework generates the necessary dialog box to populate the data contract.

The following code contains the entry point of the service:

```
[SysEntryPointAttribute(true)]
public void runOperation(PrimeNumberRange data)
{
    PrimeNumbers primeNumbers;

    // Threads mainly take effect while running in the batch framework utilizing either
    // reliable asynchronous or scheduled batch

    int i, start, end, blockSize, threads = 8;
    PrimeNumberRange subRange;
    start = data.parmStart();
    end = data.parmEnd();
    blockSize = (end - start) / threads;
    delete_from primeNumbers;
    for (i = 0; i < threads; i++)
    {
        subRange = new PrimeNumberRange();
        subRange.parmStart(start);
        subRange.parmEnd(min(start + blockSize, end));
        subRange.parmLast(i == threads - 1);
```

```
        this.findPrimes(subRange);
        start += blockSize + 1;
    }
}
```

The next sample is a method that executes differently depending on the operation mode that you chose.

Note If the method is executed in reliable asynchronous mode or scheduled batch mode, this sample also showcases a bundling pattern that was discussed earlier in the "Batch bundling" section.

```
[SysEntryPointAttribute(false)]
public void findPrimes(PrimeNumberRange range)
{
    BatchHeader batchHeader;
    SysOperationServiceController controller;
    PrimeNumberRange dataContract;
    if (this.isExecutingInBatch())
    {
        ttsBegin;
        controller = new SysOperationServiceController('PrimeNumberService',
'findPrimesWorker');
        dataContract = controller.getDataContractObject('range');

        dataContract.parmStart(range.parmStart());
        dataContract.parmEnd(range.parmEnd());
        dataContract.parmLast(range.parmLast());

        batchHeader = this.getCurrentBatchHeader();
        batchHeader.addRuntimeTask(controller, this.getCurrentBatchTask().RecId);
        batchHeader.save();
        ttsCommit;
    }
    else
    {
        this.findPrimesWorker(range);
    }
}
```

Last, but not least, is the method that does the actual work:

```
private void findPrimesWorker(PrimeNumberRange range)
{
    PrimeNumbers primeNumbers;
    int i;
    int64 time;

    for (i = range.parmStart(); i <= range.parmEnd(); i++)
    {
        if (this.isPrime(i))
        {
            primeNumbers.clear();
```

```
            primeNumbers.PrimeNumber = i;
            primeNumbers.insert();
        }
    }

    if (range.parmLast())
    {
        primeNumbers.clear();
        primeNumbers.PrimeNumber = -1;
        primeNumbers.insert();
    }
}
```

The following code contains a job that runs the prime number example in all four execution modes:

```
static void generatePrimeNumbers(Args _args)
{
    SysOperationServiceController controller;
    int i, ticks, ticks2, countOfPrimes;
    PrimeNumberRange dataContract;
    SysOperationExecutionMode executionMode;
    PrimeNumbers output;

    <… Dialog code to demo the execution modes …>

    executionMode = getExecutionMode();
    controller = new SysOperationServiceController('PrimeNumberService', 'runOperation',
executionMode);
    dataContract = controller.getDataContractObject('data');

    dataContract.parmStart(1000000);
    dataContract.parmEnd(1500000);
    delete_from output;
    ticks = System.Environment::get_TickCount();
    controller.parmShowDialog(false);
    controller.startOperation();

    <… Code to show execution times for demo purposes …>
}
```

Executing this code four times in all four execution modes produces the following results:

- **Synchronous** 35,658 prime numbers found in 44.74 seconds. However, the user could not continue working during this time.
- **Asynchronous** 35,658 prime numbers found in 46.93 seconds, but the client was responsive and the user could continue working.
- **Reliable asynchronous** 35,658 prime numbers found in 16.16 seconds by using parallel processing and starting the batch jobs immediately (as mentioned earlier in this section). This execution mode is only running the job on the batch server, but it is not entirely similar to a batch job. The jobs appear in the Batch Job form only temporarily. Another key difference is that even though reliable asynchronous mode uses the batch framework as a vehicle, reliable

asynchronous mode is not bound to the Available Threads setting that you can set in the Server Configuration form. So long as the server has resources, it will continue processing reliable asynchronous jobs in parallel and start processing new jobs as well. If you start too many jobs, you might overload your server; on the other hand, it allows programming models to use multicore systems efficiently.

- **Scheduled batch** 35,658 prime numbers found in 31.78 seconds. (The batch job did not start immediately, which caused the difference in execution time between scheduled batch mode and reliable asynchronous mode.)

Also, you need to ensure that there are sufficient CPU resources left to service the regular user load. It is usually a good idea to separate the batch workload from the regular user workload.

The SysOperation framework offers an additional way of parallelizing the workload through business logic. For example, you could build a wrapper class that performs multiple asynchronous business calls. Suppose that your wrapper class invoices all orders of a certain business account. You could build a dialog box that allows the user to select one or more customer accounts to invoice. The logic itself then performs one service call for each customer account. Note, however, that these calls might overload your server resources if not used with care. The following code is a modified version of the previous example to show what this code might look like.

Note In practice, you would use a dialog box to define your execution parameters.

```
// In practice, this wrapper should be a class and be called through a menu item in the
// appropriate execution mode.
static void generatePrimeNumbersAsyncCallPattern(Args _args)
{
    SysOperationServiceController controller;
    int i, primestart, primeend,blockSize, threads = 8,countOfPrimes,ticks,ticks2;
    PrimeNumberRange subRange;
    PrimeNumberRange dataContract;
    PrimeNumbers output;

    primestart = 1000000;
    primeend = 1500000;

    blockSize = (primeend - primestart) / threads;

    delete_from output;

    ticks = System.Environment::get_TickCount();

    for (i = 0; i < threads; i++)
    {
        controller = new SysOperationServiceController('PrimeNumberServiceAsyncCallPattern',
        'runOperation', SysOperationExecutionMode::ReliableAsynchronous);
        dataContract = controller.getDataContractObject('data');

        dataContract.parmStart(primestart);
        dataContract.parmEnd(min(primestart + blockSize, primeend));
```

```
        dataContract.parmLast(i == threads - 1);

        controller.parmShowDialog(false);
        controller.startOperation();

        primestart += blockSize + 1;
    }

    <… Code to show execution times for demo purposes …>
}
```

Patterns for checking to see whether a record exists

Depending on the pattern that you use, checking to see whether a record exists can result in excessive calls to the database.

The following code shows an incorrect example of how to determine whether a certain record exists. For each record that is fetched in the outer loop, another select statement is passed to the database to find a particular entry in the WMSJournalTrans table. If the WMSJournalTable table has 10,000 rows, the following logic would cause 10,001 queries to the database:

```
static void existingJournal()
{
    WMSJournalTable     wmsJournalTable = WMSJournalTable::find('014119_117');
    WMSJournalTable     wmsJournalTableExisting;
    WMSJournalTrans     wmsJournalTransExisting;

    boolean recordExists()
    {
        boolean foundRecord;
        foundRecord = false;

        while select JournalId from wmsJournalTableExisting
            where wmsJournalTableExisting.Posted    == NoYes::No
        {
            select firstonly wmsJournalTransExisting
                where wmsJournalTransExisting.JournalId         ==
wmsJournalTableExisting.JournalId    &&
                      wmsJournalTransExisting.InventTransType   ==
wmsJournalTable.InventTransType      &&
                      wmsJournalTransExisting.InventTransRefId  ==
wmsJournalTable.InventTransRefId;
            if (wmsJournalTransExisting)
                foundRecord = true;
        }
        return foundRecord;
    }

    if (recordExists())
       info('Record Exists');
    else
       info('Record does not exist');
}
```

The following example shows a better pattern that produces far less overhead. This pattern results in only one query and one round-trip to the database:

```
static void existingJournal()
{
    WMSJournalTable     wmsJournalTable = WMSJournalTable::find('014119_117');
    WMSJournalTable     wmsJournalTableExisting;
    WMSJournalTrans     wmsJournalTransExisting;

    boolean recordExists()
    {
        boolean foundRecord;
        foundRecord = false;

        select firstonly wmsJournalTransExisting
        join wmsJournalTableExisting
        where wmsJournalTransExisting.JournalId         ==
        wmsJournalTableExisting.JournalId    &&
        wmsJournalTransExisting.InventTransType   ==
        wmsJournalTable.InventTransType      &&
        wmsJournalTransExisting.InventTransRefId  ==
        wmsJournalTable.InventTransRefId &&
        wmsJournalTableExisting.Posted    == NoYes::No;

        if (wmsJournalTransExisting)
                foundRecord = true;

        return foundRecord;
    }

    if (recordExists())
       info('Record Exists');
    else
       info('Record does not exist');
 }
```

Run a query only as often as necessary

Often, the same query is executed repeatedly. Even if caching reduces some of the overhead, repeatedly executing the same query sometimes can have a significant impact on performance. But there are ways that you can easily avoid these performance problems. Usually, they are caused by *find* methods that are called repeatedly—either within loops or within an *exists* method. The following example shows a loop that makes repeated calls to the *CustParameters::find* method:

```
static void doOnlyNecessaryCalls(Args _args)
{
    LedgerJournalTrans ledgerJournalTrans;
    LedgerJournalTable ledgerJournalTable = LedgerJournalTable::find('000242_010');
    Voucher            voucherNum = '';

    while select ledgerJournalTrans
        order by JournalNum, Voucher, AccountType
        where ledgerJournalTrans.JournalNum == ledgerJournalTable.JournalNum
           && (voucherNum == '' || ledgerJournalTrans.Voucher == voucherNum)
```

```
    {
         // Potential unecessary cache lookup and method call if loop returns multiple rows

         ledgerJournalTrans.PostingProfile = CustParameters::find().PostingProfile;

         // Additional code doing some work…
    }
}
```

The recurring calls to *CustParameters::find* always return the same results. Even if the result is cached, these calls produce overhead. To optimize performance, you can move the call outside the loop, preventing repeated calls.

```
static void doOnlyNecessaryCallsOptimized(Args _args)
{
    LedgerJournalTrans ledgerJournalTrans;
    LedgerJournalTable ledgerJournalTable = LedgerJournalTable::find('000242_010');
    Voucher            voucherNum = '';
    CustPostingProfile postingProfile = CustParameters::find().PostingProfile;

    while select ledgerJournalTrans
        order by JournalNum, Voucher, AccountType
        where ledgerJournalTrans.JournalNum == ledgerJournalTable.JournalNum
           && (voucherNum == '' || ledgerJournalTrans.Voucher == voucherNum)
    {
         // No unecessary cache lookup and method call if loop returns more than 1 row

         ledgerJournalTrans.PostingProfile = postingProfile;

         // Additional code doing some work…
    }
}
```

When to prefer two queries over a join

For certain queries, it is difficult or almost impossible to create an effective index. This mainly occurs if an *OR* operator (or ||) is used on multiple columns.

The following example typically triggers an index join in SQL Server, which is potentially less effective than a direct lookup:

```
static void TwoQueriesSometimesBetterThenOne(Args _args)
{
    InventTransOriginId       inventTransOriginId = 5637201031;
    InventTransOriginTransfer inventTransOriginTransfer;

    // Note: Only one condition can be true at any time

    select firstonly inventTransOriginTransfer
         where inventTransOriginTransfer.IssueInventTransOrigin   == inventTransOriginId
            || inventTransOriginTransfer.ReceiptInventTransOrigin == inventTransOriginId;

    info(int642str(inventTransOriginTransfer.RecId));
}
```

Using two queries might cause an additional round-trip, but ideally, the following code produces only one. In addition, the first and second queries are efficient direct-clustered and direct-index lookups. In practice, you would need to test this code to ensure that it outperforms the earlier example in your scenario.

```
static void TwoQueriesSometimesBetterThenOneOpt(Args _args)
{
    InventTransOriginId inventTransOriginId = 5637201031;
    InventTransOriginTransfer inventTransOriginTransfer;

    select firstonly inventTransOriginTransfer
        where inventTransOriginTransfer.IssueInventTransOrigin == inventTransOriginId;

    info(int642str(inventTransOriginTransfer.RecId));

    if(!inventTransOriginTransfer.RecId)
    {
        select firstonly inventTransOriginTransfer
            where inventTransOriginTransfer.ReceiptInventTransOrigin == inventTransOriginId;

        info(int642str(inventTransOriginTransfer.RecId));
    }
}
```

Indexing tips and tricks

Included columns is a new feature that helps you create optimized indexes. With included columns, it is easier, for example, to create covering indexes for queries with limited field lists or for queries that aggregate data. For more information about covering indexes and indexes with included columns, see "Index with Included Columns" on MSDN at *http://msdn.microsoft.com/en-us/library/ms190806.aspx*.

To create an index with included columns, set the *IncludedColumn* property on the index to *Yes*, as shown in Figure 13-10.

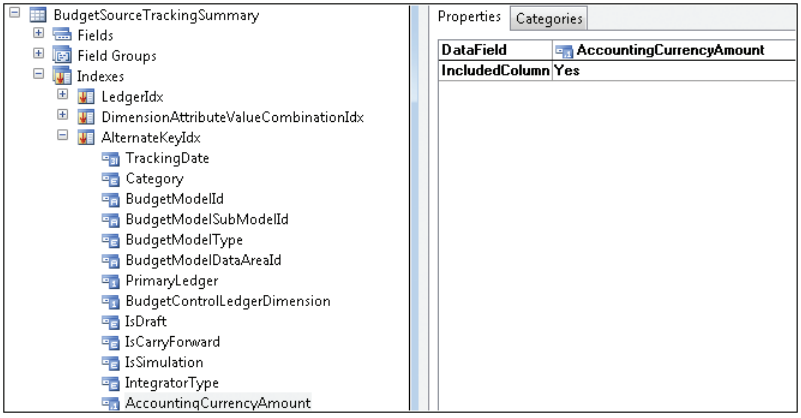


FIGURE 13-10 *IncludedColumn* property on an index.

Another lesser-known feature is that if you add the *dataAreaId* field to the key columns of an index, the AOS will not add it as the leading column in the index, which allows better optimization of certain queries. For example, queries that don't include the *dataAreaId* and use direct SQL trigger an index scan if the *dataAreaId* is the leading column of an index when the index is used. In general, you should use this feature only if you notice that the *dataAreaId* is not in the query and SQL Server is performing an index scan because of that. However, this is not recommended unless it is necessary. If you use this technique, you should always create a new index for that purpose.

When to use *firstfast*

The *firstfast* hint adds *OPTION(FAST n)* to a SQL Server query and causes SQL Server to prefer an index that is good for sorting because the query returns the first rows as quickly as possible.

```
select firstfast salestable // results in
SELECT <FIELDLIST> FROM SALESTABLE OPTION(FAST 1)
```

Note If you are sorting fields from more than one table, *OPTION(FAST n)* might not produce the performance improvement you want.

This keyword is used automatically for grids on forms and can be enabled on the data sources of AOT queries. As beneficial as this keyword can be—for example, on list pages that are supported by AOT queries—it can produce a performance penalty on queries in general because it causes SQL Server to optimize for sorting instead of for fastest execution time. If you see the *firstfast* hint in a query that is running slowly, try disabling it and then check the response time. The Export Letter of Credit/Import Collection form is an example of where this setting makes a difference. In the AOT, navigate to Forms\BankLCExportListPage\Data Sources\BankLCExportListPage\Data Sources\SalesTable (SalesTable). On this list page, the *FirstFast* property is set to *No*; however, performance will improve by setting it to *Yes*.

Optimize list pages

You can experiment with a set of optimizations to improve the performance of list pages. Often, list page queries are complex and span multiple data sources. Sorting joined result sets can lead to a performance penalty. To optimize performance, try reducing sorting. For example, reducing sorting can benefit performance for the Contacts form. The query *smmContacts_NoFilter* (Forms/smmContactsListPage/DataSources/smmContacts_NoFilter) specifies two tables in its *Order by* clause. To optimize performance, you can sort by *ContactPerson.ContactForParty* only.

You can also optimize list page performance by working with the *FirstFast* and *OnlyFetchActive* properties. Both options are described in detail earlier in this chapter.

Aggregate fields to reduce loop iterations

Instead of iterating and aggregating within X++ logic, you can often aggregate within the code to save loop iterations and round-trips to the database. The number of loop iterations that you can eliminate depends mainly on the fields on which the aggregation takes place and how many rows can be aggregated. There are instances when you might want to add some values within your code only based on certain conditions.

The following example compares set-based operations and aggregation with row-based operations:

```
// In practice, you should use static server methods to access data on the server.

public static void main(Args _args)
{
    TransferToSetBased ttsb;
    RecordInsertList   ril = new RecordInsertList(tableName2id("TransferToSetBased"));
    Counter            i;
    Counter            tc;
    int                myAggregate = 0;
    int                my2ndAggregate;

    // Reset table.

    delete_from ttsb;

    // Populate line-based.

    tc = WinAPI::getTickCount();
    for(i=0;i<=1000;i++)
    {
        ttsb.clear();
        ttsb.Iterate=i;
        ttsb.Change=1;
        ttsb.Aggregate=5;
        ttsb.insert();
    }

    // Data populated 1000 records, 1000 round-trips.

    for(i=1001;i<=2000;i++)
    {
        ttsb.clear();
        ttsb.Iterate=i;
        ttsb.Change=1;
        ttsb.Aggregate=5;
        ril.add(ttsb);
    }
    ril.insertDatabase();

    // Data populated 1000 records, many fewer round-trips.
    // Based on buffer size. About 20-150 inserts per round-trip.

    ttsBegin;
    while select forupdate ttsb where ttsb.Iterate > 1000
```

```
{
    if(ttsb.Iterate >= 1100 && ttsb.Iterate <= 1300)
    {
        ttsb.Change = 10;
        ttsb.update();
        myAggregate += ttsb.Aggregate;
    }
    else if(ttsb.Iterate >= 1301 && ttsb.Iterate <= 1500)
    {
        ttsb.Change = 20;
        ttsb.update();
        my2ndAggregate += ttsb.Change;
    }
    else if(ttsb.Iterate >= 1501 && ttsb.Iterate <= 1700)
    {
        ttsb.Change = 30;
        ttsb.update();
        myAggregate += ttsb.Aggregate;
    }

    if(ttsb.Iterate > 1900)
        break;
}
ttsCommit;

// While loop does 1-900 fetches. Does 600 single update statements.
// Above logic set-based and using aggregation results in 6 queries to the database.

update_recordSet ttsb setting change = 10 where ttsb.Iterate >= 1100 && ttsb.iterate <=
        1300;
update_recordSet ttsb setting change = 20 where ttsb.Iterate >= 1301 && ttsb.Iterate <=
        1500;
update_recordSet ttsb setting change = 30 where ttsb.Iterate >= 1501 && ttsb.Iterate <=
        1700;

select sum(Aggregate) from ttsb where ttsb.Iterate >= 1100 && ttsb.Iterate <= 1300;
myAggregate = 0;
myAggregate = ttsb.Aggregate;

select sum(Change) from ttsb where ttsb.Iterate >= 1301 && ttsb.Iterate <= 1500;
my2ndAggregate = ttsb.Change;

select sum(Aggregate) from ttsb where ttsb.Iterate >= 1501 && ttsb.Iterate <= 1700;
myAggregate += ttsb.Aggregate;

}
```

Performance monitoring tools

Without a way to monitor the execution of your application logic, you implement features almost blindly with regard to performance. Fortunately, the Microsoft Dynamics AX Development Workspace contains a set of easy-to-use tools to help you monitor client/server calls, database activity, and

application logic. These tools provide good feedback on the feature being monitored. The feedback is integrated directly with the Development Workspace, making it possible for you to jump directly to the relevant X++ code.

Microsoft Dynamics AX Trace Parser

The Microsoft Dynamics AX Trace Parser consists of a user interface and data analyzer that is built on SQL Server 2008 and the Event Tracing for Windows (ETW) framework. The Microsoft Dynamics AX Trace Parser has been significantly improved in Microsoft Dynamics AX 2012, with new features and enhanced performance and usability. The performance overhead for running a single trace is comparatively low. With ETW, you can conduct tracing with system overhead of approximately 4 percent.

Only users with administrative privileges, users in the Performance Log Users group, and services running as LocalSystem, LocalService, and NetworkService can enable trace providers.

To use the Microsoft Dynamics AX Tracing Cockpit in the client, a user must be either in the Administrators or Performance Log Users group. The same is true for users who use Windows Performance Monitor. Additionally, the user must have write access to files in the folder that stores the results of the trace.

The Trace Parser enables rapid analysis of traces to find the longest-running code, the longest-running SQL query, the highest call count, and other metrics that are useful in debugging a performance problem. In addition, it provides a call tree of the code that was executed, allowing you to gain insight into unfamiliar code quickly. It also provides the ability to jump from the search feature to the call tree so that you can determine how the problematic code was called.

The Trace Parser is included with Microsoft Dynamics AX 2012 and is also available as a free download from Partner Source and Customer Source. To install the Trace Parser, run the Microsoft Dynamics AX 2012 Setup program and navigate to Add or Modify Components > Developer Tools > Trace Parser.

New Trace Parser features

Microsoft Dynamics AX 2012 includes several new features for Trace Parser, which can help you understand a performance problem quickly.

- **Monitor method calls** If you right-click a line in X++/RPC view, and then click Jump To Non-Aggregated View, you can view, information such as whether all calls to the method took the same amount of time or if one call was an outlier. The same function is available in the SQL view.
- **Monitor client sessions** If there was an RPC call between the client and the server in either Non-Aggregated view or Call Tree view, you can right-click the line containing the call, and then click Drill Through To Client Session. This feature also works for RPC calls between the server and the client.

- **Jump between views** If you want to jump from Call Tree view to Non Aggregated X++/RPC view, you can right-click, and then select the option you want.
- **Monitor events** In either Non Aggregated X++/RPC view or Call Tree view, you can select two or more events while holding down the Ctrl key and then right-click and select Show Time Durations Between Events. This is extremely useful for monitoring and troubleshooting asynchronous events.
- **Look up table details** Under View, you can click Table Details to look up table details within Microsoft Dynamics AX. A Business Connector .NET connection is required for this functionality, just like the code lookup functionality.
- **Compare traces** Under View, you can click Trace Comparison, which opens a form where you can compare two traces.

Before tracing

Before taking a trace, run the process that you want to trace at least once to avoid seeing metadata loading in the trace file. This is called tracing in a warm state and is recommended because it helps you to focus on the real performance issue and not on metadata loading and caching. Then you can prepare everything so that the amount of time between starting the trace and executing the process you want to trace is as short as possible.

In Microsoft Dynamics AX 2009, you had to set tracing options in multiple places. In Microsoft Dynamics AX 2012, there are only three places to set options. In addition, there is only one trace file for both the client and the server.

You can start a trace in three ways:

- From the Tracing Cockpit in the Microsoft Dynamics AX 2012 client
- From Windows Performance Monitor
- Through code instrumentation

The following sections describe each method in detail.

Start a trace through the client

As mentioned earlier, you must be logged on as an administrator to use the Tracing Cockpit. Table 13-2 describes the options that are available in the Tracing Cockpit.

TABLE 13-2 Options in the Tracking Cockpit.

Element	Description
Start Trace	Start tracing after you specify the location where you want to store the trace file.
Stop Trace	Stop tracing and finish writing the information to your trace file.
Cancel Trace	Stop the trace without saving information to the trace file.
Open Trace	Open the trace file in Trace Parser.

Element	Description
Collect Server Trace	Collect both client and server data.
Circular Logging	Specify a file size and keep logging information until you click Stop. If you select this option, data is overwritten, so you get the latest data in the file. This option is new for Microsoft Dynamics AX 2012 and is especially effective if you want to trace processes that run longer than, for example, 10 minutes. You can use this feature to capture a trace in the middle of the execution of a long-running process.
Bind Parameters	Allow users to get the actual values that are passed to SQL Server instead of the parameterized queries. This option is turned off by default because it potentially collects confidential information.
Detailed Database	Collect information about the number of rows fetched and the time it took to fetch those rows.
RPC	Collect information about the number of RPC calls that are being made.
SQL	Collect the SQL statements that the AOS passes to SQL Server.
TraceInfo	Show information about what process logged the event.
TTS	Log the *ttsBegin*, *ttsCommit*, and *ttsAbort* statements.
XPP	Log the X++ calls that are being made.
XPP Marker	Copy markers that are added during the trace to the trace file.
Client Access	Collect information about which forms were opened and closed and which buttons were clicked.
XPP Parameter Info	Collect the parameters passed to X++ methods. This option is turned off by default because it potentially collects confidential information.

To start a trace from the Tracing Cockpit, do the following:

1. In the Microsoft Dynamics AX 2012 client, open the Development Workspace by pressing Ctrl+Shift+W.
2. On the Tools menu, click Tracing Cockpit (Figure 13-11).
3. Set the options for your trace. For example, if you only want to collect a client trace, clear the Collect Server Trace check box.
4. Bring your process to a warm state (as described earlier) and then click Start Trace.
5. Choose a location in which to save your trace file.
6. Execute your process, and then click Stop Trace.
7. Click Open Trace to open the trace file in the Trace Parser.

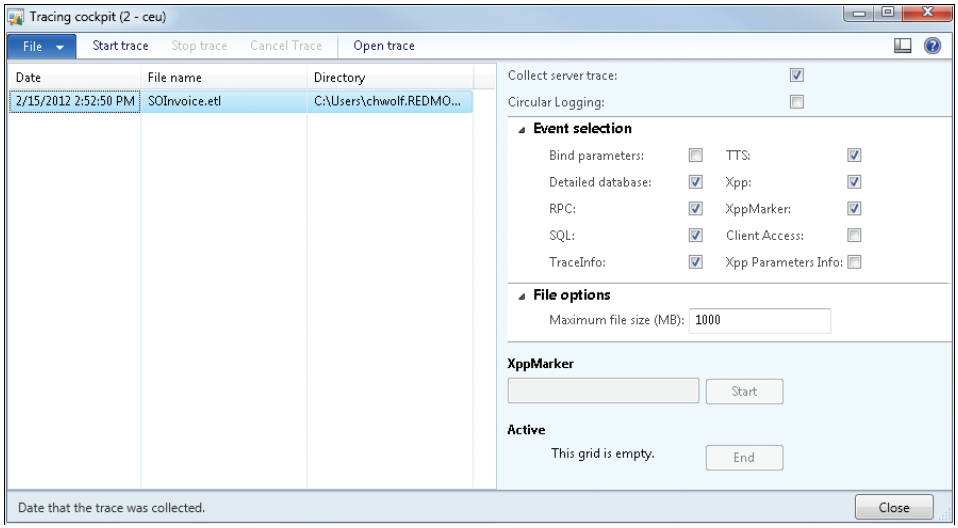


FIGURE 13-11 The Tracing Cockpit.

Start a trace through Windows Performance Monitor

To start a trace in Windows Performance Monitor, do the following:

1. On the Start menu, click Run, and then type **perfmon**.
2. Expand Data Collector Sets.
3. Right-click User Defined, and then click New > Data Collector Set.
4. Select Create Manually, and then click Next.
5. Select Event Trace Data, and then click Next.
6. Next to Providers, click Add, and then In the Event Trace Providers form, select Microsoft-DynamicsAX-Tracing, and then click OK.

Note If you use Windows Performance Monitor, by default, all events are traced, including events that might collect confidential information. To prevent this, click Edit, and then select only the events necessary. The events that might collect confidential information are noted in their descriptions.

7. Click Next, and then note the root directory that your traces are stored in.
8. Click Next to change the user running the trace to an Administrative user, and then click Finish.
9. In the right pane of Windows Performance Monitor, right-click the newly created data collector set, and click Properties.

10. In the Properties window, click the Trace Buffers tab and modify the default buffer settings. The default buffer settings do not work well for collecting Microsoft Dynamics AX event traces because large numbers of events can be generated in a short time and fill the buffers quickly. Change the following settings as specified and leave the rest set to the default:
    - Buffer Size: 512 KB
    - Minimum Buffers: 60
    - Maximum Buffers: 60
11. To start tracing, click the data collector set in the left pane, and then click Start.

Start a trace through code instrumentation

You can use the *xClassTrace* class from the Tracing Cockpit to start and stop a trace. To trace the Sales Form letter logic, see the following sample in \Classes\SalesFormLetter:

```
// Add
xClassTrace xCt = new xClassTrace();

// to the variable declaration.
// …code…

        if (salesFormLetter.prompt())
        {
            xClassTrace::start("c:\\temp\\test1.etl");
            xClassTrace::logMessage("test1");
            xCt.beginMarker("marker"); // Add markers at certain points of a trace to
                                       // increase trace readability. You can add
                                       // multiple markers per trace.

            salesFormLetter.run();

            xCt.endMarker("marker");
            xClassTrace::stop();

            outputContract  = salesFormLetter.getOutputContract();
            numberOfRecords = outputContract.parmNumberOfOrdersPosted();
        }

// …code…
```

In the call to *xClassTrace::start*, you can use multiple parameters to specify the events to trace or whether you want to use circular logging, among other things. To find out which keyword equals which parameter, put a breakpoint in the class *SysTraceCockpitcontroller\startTracing* and start a trace from the Tracing Cockpit with various events selected.

Import a trace

To import a trace, open the Microsoft Dynamics AX Trace Parser, and then click Import Trace. (You can also use the Open Trace form to import a trace file.) It is possible to import multiple trace files at once.

Analyze a trace

After you load the trace files into the Trace Parser, you can analyze your trace files through built-in views.

When you open a trace from the Overview tab, you see a summary that gives you a high-level understanding of where the most time is spent within the trace.

On the Overview tab, select a session. If you took the trace, select your session. If you received the trace file from someone else, select the session of the person who took the trace. When you select a session, you'll see an overview similar to Figure 13-12, but for that session only. To return to the summary for all sessions, select the Show Summary Across All Sessions check box.

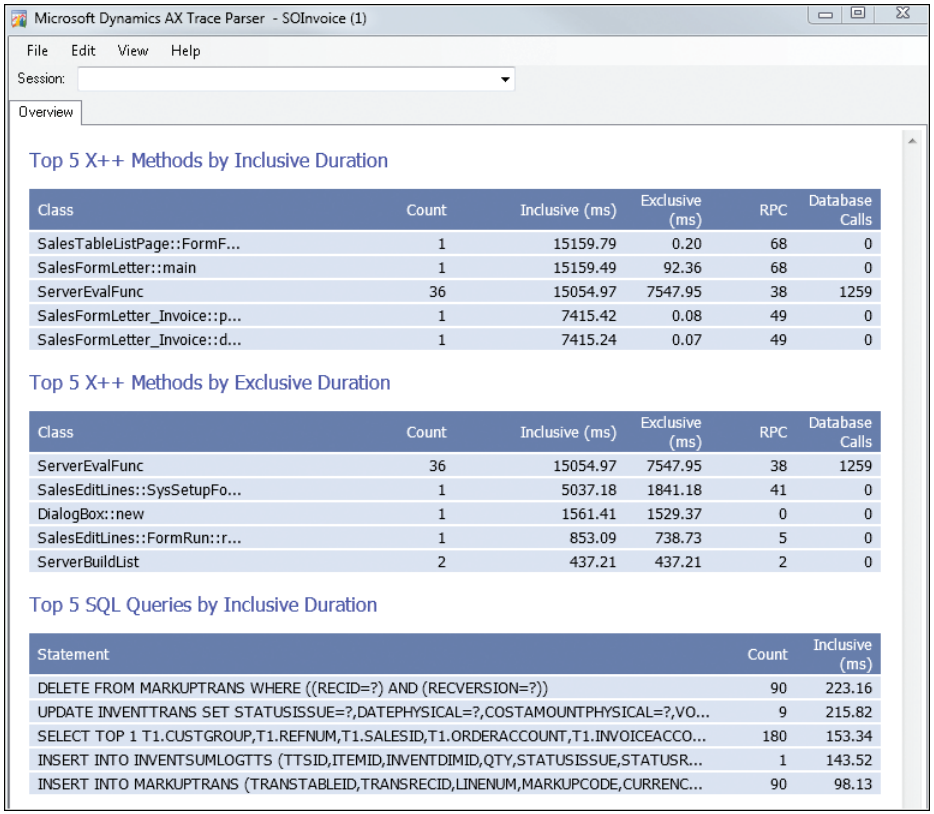


FIGURE 13-12 Trace overview.

After selecting a session in the drop-down list, you can search and review the trace through the X++ methods and RPC calls or the SQL queries, or you can review the call tree of the session. It's best to start looking for quick improvements by sorting by total exclusive duration. Then, break the process down by sorting by total inclusive duration for detailed tuning. You can jump to the Call Tree view from the X++ methods and RPC calls and from the SQL view.

Use the X++/RPC view to understand patterns in your trace, as shown in Figure 13-13.

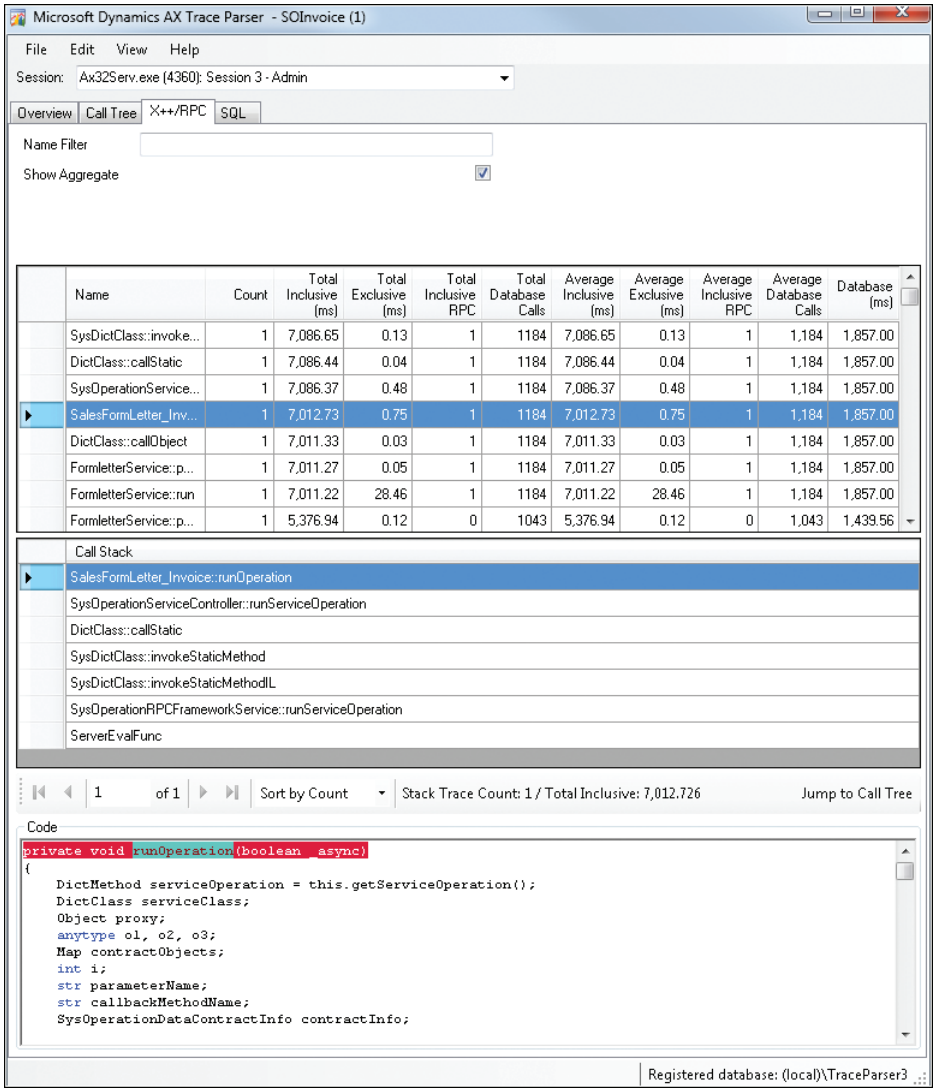


FIGURE 13-13 X++/RPC view.

SQL view (see Figure 13-14) gives you a quick overview of which queries were executed and how long the execution and data retrieval took.

Note Execution time and row retrieval time are measured separately.

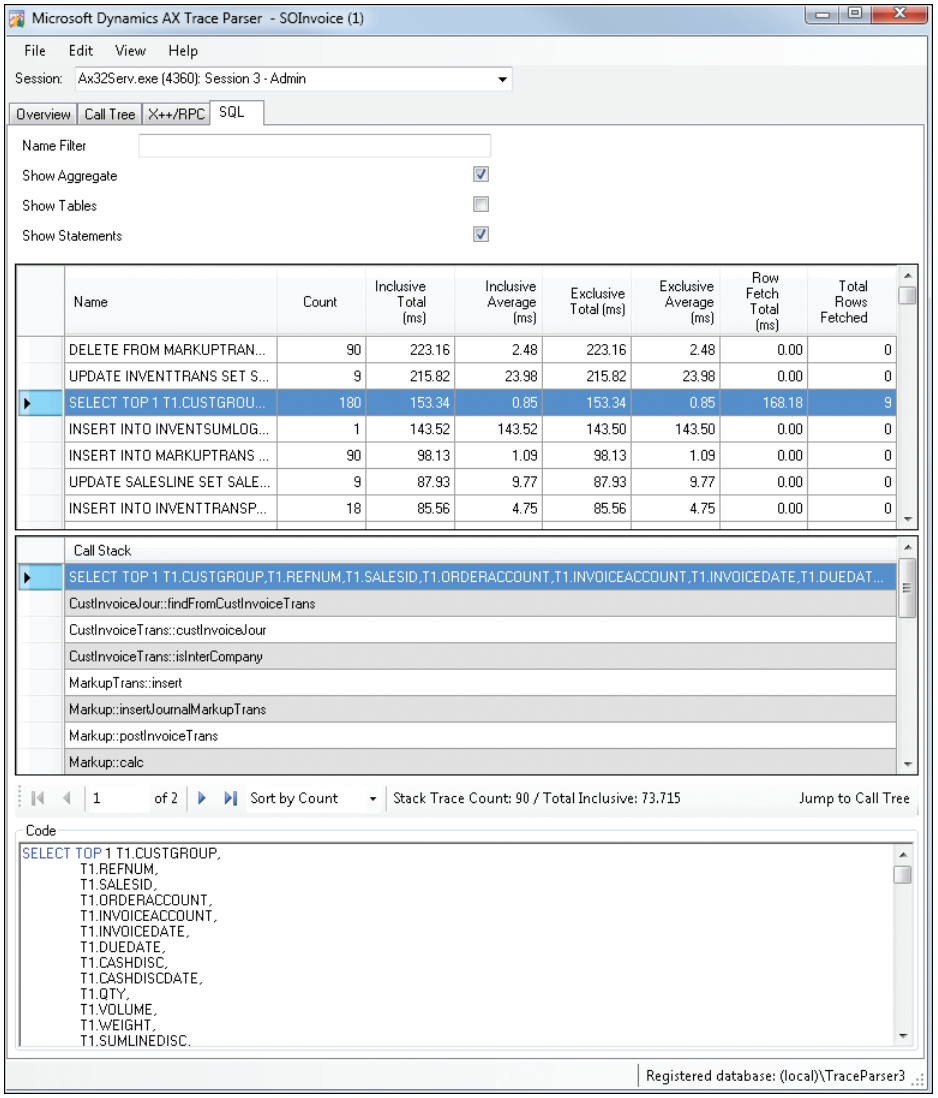


FIGURE 13-14 SQL view.

Call Tree view (see Figure 13-15) is particularly helpful for identifying expensive loops and other costly patterns.

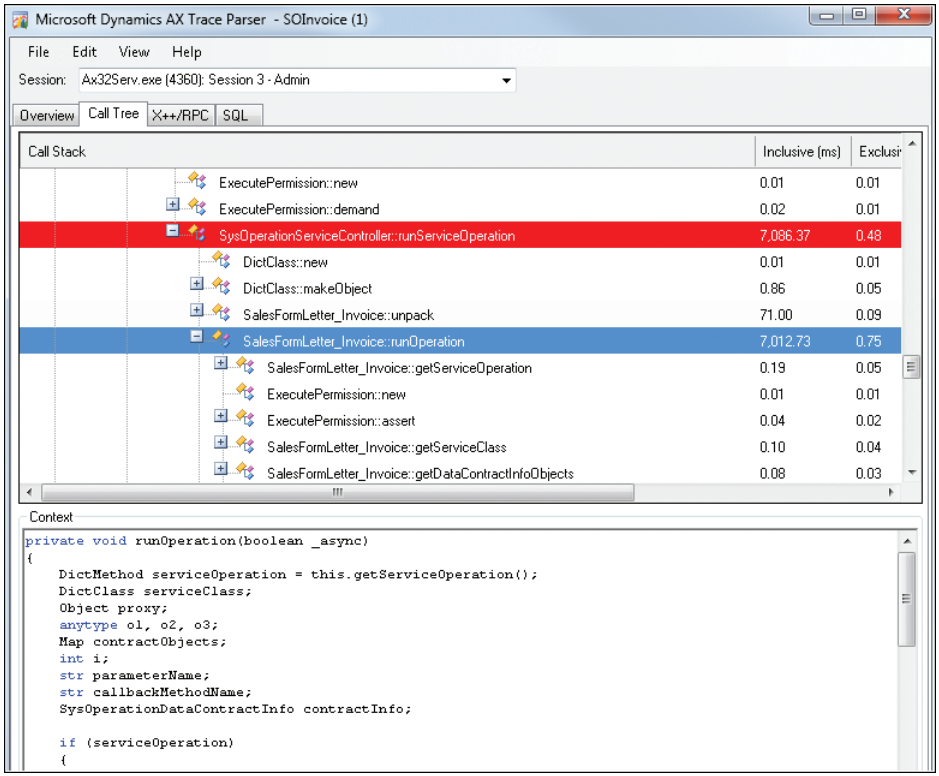


FIGURE 13-15 Call Tree view.

Troubleshoot tracing

This section provides information about how to troubleshoot a few of the common issues with tracing.

Tracing won't start If tracing doesn't start, make sure that the user who is running the trace is a member of the Administrators or Performance Log Users group.

Tracing causes performance problems If you run a trace from a client that is located on an AOS, you will get one trace file. If the client is not on the AOS, you will get two files: one on the client computer and one on the AOS. If you run more than one client tracing session simultaneously, the system will slow down because tracing is processing- and space-intensive in this situation. It is recommended that you not turn on tracing on an AOS instance that is supporting a workload of multiple clients.

Trace doesn't produce meaningful data If X++ code is running as CIL, a trace might not produce meaningful results. Table 13-3 lists scenarios that might cause tracing problems and describes possible mitigations.

TABLE 13-3 Troubleshooting tracing for X++ code running as CIL.

Scenario	Mitigation
X++ code is traversed into CIL by means of *RunAs*	In the Development Workspace, click Tools > Options. On the Development tab, clear the Execute Business Operations In CIL check box.
Services that are called from outside Microsoft Dynamics AX or services in the AxClient group	Often it is effective to write a small test job or class to execute the service from within Microsoft Dynamics AX. If for some reason, this is not an option, utilize Microsoft Visual Studio profiling to trace the service.
Batch jobs run in CIL	Execute the code outside the batch framework. Try to limit the length of the operation; for example, by limiting the operation to a small number of tasks that can be processed in a few minutes. If this is not possible, you can use Visual Studio profiling, which is described at the end of this chapter.

Monitor database activity

You can also trace database activity when you're developing and testing Microsoft Dynamics AX application logic.

You can enable tracing on the SQL tab of the Options dialog box (in the AOT, on the Tools menu, click Options). You can trace all Transact-SQL statements or just the long-running queries, warnings, and deadlocks. Transact-SQL statements can be traced to the Infolog, a message window, a database table, or a file. If statements are traced to the Infolog, you can use the context menu to open the statement in the SQL Trace dialog box, in which you can view the entire statement and the path to the method that executed the statement.

Note You should not use this feature except for long-term monitoring of long-running queries. Even then, you should use this feature carefully because it adds overhead to the system.

From the SQL Trace dialog box, you can copy the statement and, if you're using SQL Server 2008, open a new query window in SQL Server Management Studio (SSMS) and paste in the query. If the Microsoft Dynamics AX run time uses placeholders to execute the statement, the placeholders are shown as question marks in the statement. You must replace these with variables or constants before the queries can be executed in SQL Server Query Analyzer. If the run time uses literals, the statement can be pasted directly into SQL Server Query Analyzer and executed.

When you trace SQL statements in Microsoft Dynamics AX, the run time displays only the DML statement. It doesn't display other commands that are sent to the database, such as transaction commits or isolation level changes. With SQL Server 2008 and later versions, you can use SQL Server Profiler to trace these statements by using the event classes *RPC:Completed* and *SP:StmtCompleted* in the Stored Procedures collection, and the *SQL:BatchCompleted* event in the TSQL collection, as shown in Figure 13-16.

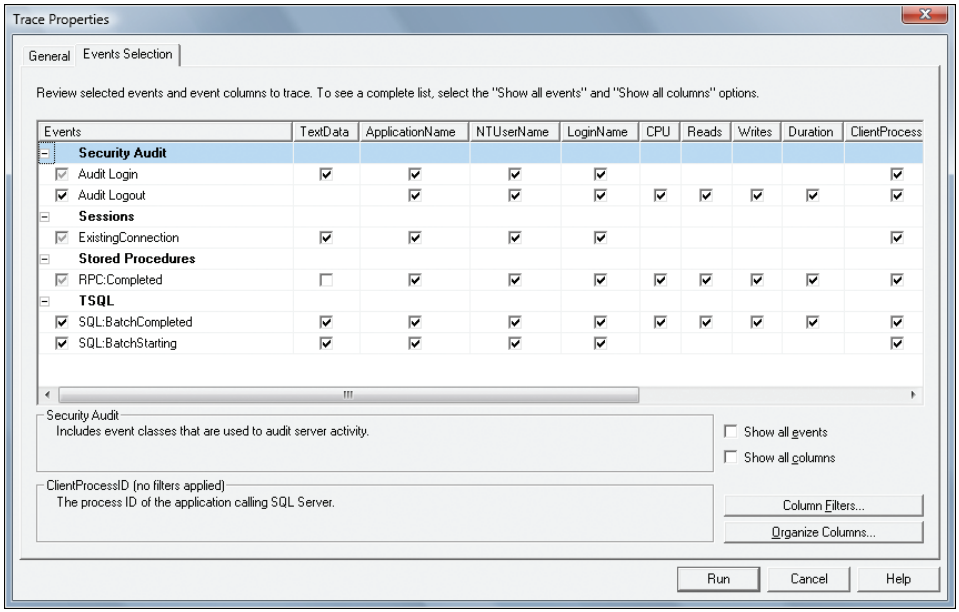


FIGURE 13-16 SQL Server Profiler trace events.

Use the SQL Server connection context to find the SPID or user behind a client session

You can use the Server Process ID (SPID) or user name for a client session to troubleshoot a wide variety of issues, such as contention or queries that run slowly. In previous versions of Microsoft Dynamics AX, the Online Users form contained a column for the SPID of client sessions. In Microsoft Dynamics AX 2012, information about user sessions can be included in the SQL Server connection context. Adding this information has a small performance overhead.

For more information, see the entry "Finding User Sessions from SPID in Dynamics AX 2012" on the Thoughts on Microsoft Dynamics AX blog (*http://blogs.msdn.com/b/amitkulkarni/archive/2011/08/10/finding-user-sessions-from-spid-in-dynamics-ax-2012.aspx*).

After applying the information from the blog entry, you can also use the following query returns session information, including the user names of Microsoft Dynamics AX users and, to some extent, the queries that they are currently running:

```
select top 20 cast(s.context_info as varchar(128)) as ci,text,query_plan,* from
sys.dm_exec_cursors(0) as ec cross apply sys.dm_exec_sql_text(sql_handle) sql_text,
sys.dm_exec_query_stats as qs cross apply sys.dm_exec_query_plan(plan_handle) as
plan_text,sys.dm_exec_sessions s
where ec.sql_handle = qs.sql_handle and ec.session_id = s.session_id order by ec.worker_time
desc
```

The client access log

You can use the client access log to track the activities of multiple users as they do their daily work. The client access log writes data to the SysClientAccessLog table. For more information about this feature, see the entry "Client Access Log" on the Microsoft Dynamics AX Performance Team blog (*http://blogs.msdn.com/b/axperf/archive/2011/10/14/client-access-log-dynamics-ax-2012.aspx*).

Visual Studio Profiler

As mentioned earlier, for certain processes, the only option for tracing might be Visual Studio Profiler. The following are high-level steps for using Visual Studio Profiler with Microsoft Dynamics AX.

Note Visual Studio Profiler is available with Visual Studio 2010 Premium and Visual Studio 2010 Ultimate editions.

1. In Visual Studio, on the Debug menu, click Options And Settings.
2. In the left pane of the Options dialog box, click Debugging, and then click Symbols, and ensure that the symbol file is loaded for the XppIL folder of the AOS that you want to profile. (The profiling tools use symbol [.pdb] files to resolve symbolic names such as function names in program binaries.)
3. On the Analyze menu, click Launch Performance Wizard to create a new performance session.
4. Accept the default setting of CPU Sampling, and point to the AOS that you want to profile, but don't start profiling right away.
5. Open Performance Explorer, right-click the top node of your session (Figure 13-17), and then click Properties.

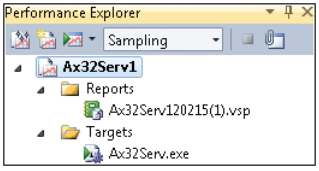


FIGURE 13-17 Performance Explorer.

6. In the Properties window, navigate to Sampling and decrease the sampling interval either to 100,000 or 1,000,000 to get better results.
7. Prepare the process that you want to profile, and then click Attach/Detach to attach to the process (for example, the AOS).
8. When you are done profiling, click Attach/Detach to detach from the AOS.

Important Don't click Stop Profiling because this will cause the AOS to stop responding.

After you finish profiling, Visual Studio generates a report that helps you understand the performance problem in detail, as shown in Figure 13-18.

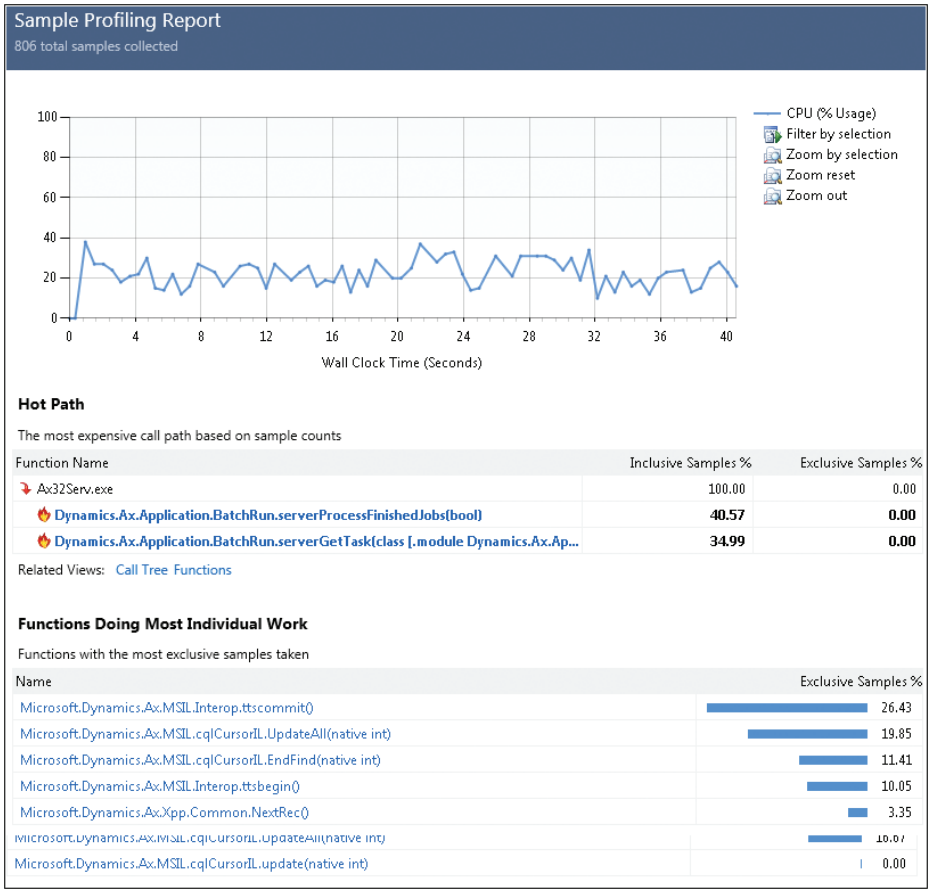


FIGURE 13-18 Profiling report.

The report offers multiple views such as Summary, Call Tree, and Functions, and it offers options to show functions that called the function you are currently reviewing. If you installed the Visual Studio tools for Microsoft Dynamics AX, you can also quickly navigate to the X++ methods identified in the report without leaving Visual Studio.

Tip The smaller the sampling interval, the better the quality of the profiling, but more data is collected.

CHAPTER 14

Extending Microsoft Dynamics AX

In this chapter

Introduction 493
The SysOperation framework 493
Comparing the SysOperation and RunBase frameworks 495
The RunBase framework 510
The extension framework 516
Eventing 520

Introduction

Microsoft Dynamics AX provides several frameworks that you can use to extend an application. In Microsoft Dynamics AX 2012, the SysOperation framework replaces the RunBase framework to provide support for business transaction jobs, such as exchange rate adjustment or inventory closing. Microsoft Dynamics AX also provides two extensibility patterns: the extension framework, which works well for developing plug-ins, and the eventing framework, which is based on eventing concepts in the Microsoft .NET Framework.

The first part of this chapter introduces the SysOperation framework and discusses an example that compares the SysOperation and RunBase frameworks. The next section provides more information about RunBase classes to help you understand existing functionality developed with the RunBase framework.

The final sections describe the extension and eventing frameworks. The extension framework reduces or eliminates the coupling between application components and their extensions. The eventing framework is new in Microsoft Dynamics AX 2012. The methods in an X++ class can raise an event immediately before they start (the *pre* event), and again after they end (the *post* event). These two events offer opportunities for you to insert custom code into the program flow with event handlers.

The SysOperation framework

You use the SysOperation framework when you want to write application logic that supports running operations interactively or by means of the Microsoft Dynamics AX batch server. This framework provides capabilities that are similar to those of the RunBase framework.

The batch framework, which is described in detail in Chapter 18, "The batch framework," has specific requirements for defining operations:

- The operation must support parameter serialization so that its parameters can be saved to the batch table.
- The operation must have a way to display a user interface that lets users modify batch job parameters. For more information about batch jobs in Microsoft Dynamics AX, see Chapter 18 and the topic "Process batch jobs and tasks," at *http://technet.microsoft.com/en-us/library/gg731793.aspx*.
- The operation must implement the interfaces needed for integration with the batch server run time.

While the RunBase framework defines coding patterns that implement these requirements, the SysOperation framework goes further by providing base implementations for many of the interfaces and classes in the patterns.

Unlike the RunBase framework, the SysOperation framework implements the Model-View-Controller (MVC) design pattern, separating presentation from business logic. For more information, see "Model-View-Controller," at *http://msdn.microsoft.com/en-us/library/ff649643.aspx*.

SysOperation framework classes

The *SysOperationServiceController* class provides several useful methods, such as the following:

- ***getServiceInfo*** Gets the service operation.
- ***getDataContractInfo*** Gets the data contracts that are used as parameters and return values for the service operation, and gets the user interface (UI) builder information for each of the data contracts.
- ***startOperation*** Makes the service call in various modes, including synchronous, asynchronous, and batch.

The *SysOperationUIBuilder* and *SysOperationAutomaticUIBuilder* classes help to create the default user interface from a definition of the data contract or from a custom form definition. You can write custom UI builders that derive from this base class to provide defaulting and validation or to raise specific events. You can override the following methods:

- ***postBuild*** Overriding this method lets you get references to the dialog box controls if the UI builder is dynamic (In other words, if the UI builder is not form-based).
- ***postRun*** Overriding this method lets you register validation methods.

SysOperation framework attributes

SysOperation attributes specify metadata for the data contracts to provide loose coupling with UI builders. The following attributes are available:

- ***DataContractAttribute*** Identifies a class as a data contract.
- ***DataMemberAttribute*** Identifies a property as a data member.
- ***SysOperationContractProcessingAttribute*** Designates a default UI builder for the data contract.
- ***SysOperationLabelAttribute***, ***SysOperationHelpTextAttribute***, and ***SysOperationDisplayOrderAttribute*** Specify the label, help text, and display order attributes, respectively, for the data member.

Comparing the SysOperation and RunBase frameworks

The SysOperation and the RunBase frameworks are designed to build applications that have operations that can run on the batch server or interactively. For an operation to run on the batch server, it must support the following:

- Parameter serialization by means of the *SysPackable* interface
- The standard run method that is defined in the *BatchRunable* interface
- The batch server integration methods found in the *Batchable* interface
- A user interface that enables and displays user input

Figure 14-1 illustrates how all operations that must run by means of the batch server must derive from either the *SysOperationController* or the *RunBaseBatch* base class.

The code examples in the following sections illustrate the basic capabilities provided by the two frameworks. These examples run an operation both interactively (by means of a dialog box) and in batch mode.

To view and use the samples on your own, import *PrivateProject_SysOperationIntroduction.xpo*, and then press Ctrl+Shift+P to view the sample code in the Projects window. You can view the following two sample classes in the *Sample_1_SysOperation_Runbase_Comparison* node:

- *SysOpSampleBasicRunbaseBatch*
- *SysOpSampleBasicController*

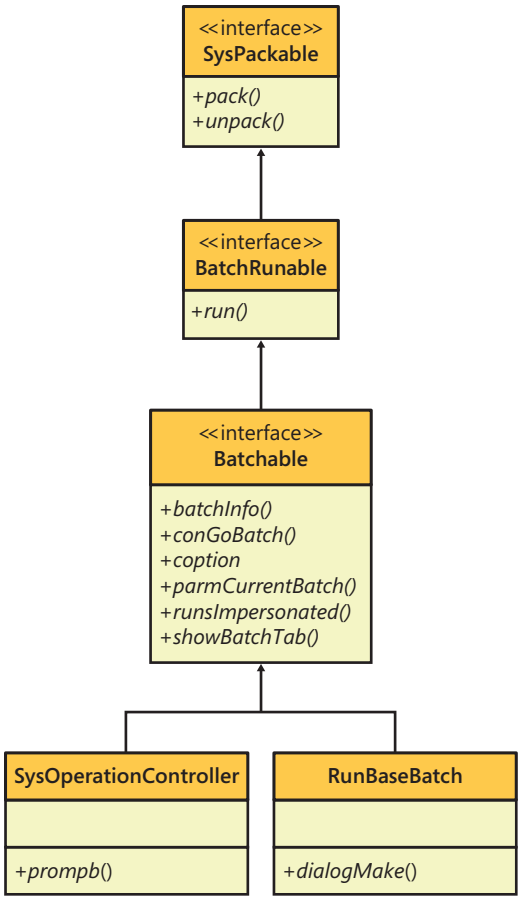


FIGURE 14-1 Derivation of operations that run on the batch server.

These classes compare the functionality of the RunBase framework to the functionality of the SysOperation framework.

Before you run the samples, you must compile the project and generate common intermediate language (CIL) for the samples.

1. In the Development Workspace, right-click the project name, and then click Compile.
2. Click Build and then click Generate Incremental CIL (or press Ctrl+Shift+F7).

RunBase example: *SysOpSampleBasicRunbaseBatch*

The simplest operation that is based on the *RunBaseBatch* base class must implement several overridden methods. Table 14-1 describes the overridden methods that are implemented in the *SysOpSampleBasicRunbaseBatch* class. Example code following the table illustrates how to use these methods.

TABLE 14-1 Method overrides for the *RunBaseBatch* class.

Method	Description
dialog	Populates the dialog box created by the base class with controls needed to get user input
getFromDialog	Transfers the contents of dialog box controls to operation input parameters
putToDialog	Transfers the contents of operation input parameters to dialog box controls
pack	Serializes operation input parameters
unpack	Deserializes operation input parameters
run	Runs the operation
description	A static description for the operation

In the override for the *classDeclaration* method that derives from *RunBaseBatch*, you must declare variables for input parameters, dialog box controls, and a macro, LOCALMACRO, that defines a list of variables that must be serialized:

```
class SysOpSampleBasicRunbaseBatch extends RunBaseBatch
{
    str text;
    int number;
    DialogRunbase       dialog;

    DialogField numberField;
    DialogField textField;

    #define.CurrentVersion(1)

    #LOCALMACRO.CurrentList
        text,
        number
    #ENDMACRO
}
```

Next, override the *dialog* method. This method populates the dialog box created by the base class with two controls that accept user input: a text field and a numeric field. The initial values from the class member variables are used to initialize the controls. Note that the type of each control is determined by the name of the extended data type (EDT) identifier:

```
protected Object dialog()
{
    dialog = super();

    textField = dialog.addFieldValue(IdentifierStr(Description255),
        text,
        'Text Property',
        'Type some text here');

    numberField = dialog.addFieldValue(IdentifierStr(Counter),
        number,
        'Number Property',
        'Type some number here');
```

```
    return dialog;
}
```

The overridden *getFromDialog* method transfers the contents of the dialog box controls to operation input parameters:

```
public boolean getFromDialog()
{
    text = textField.value();
    number = numberField.value();

    return super();
}
```

The overridden *putFromDialog* method transfers the contents of operation input parameters to dialog box controls:

```
protected void putToDialog()
{
    super();

    textField.value(text);
    numberField.value(number);
}
```

The overridden *pack* and *unpack* methods serialize and deserialize the operation input parameters:

```
public container pack()
{
    return [#CurrentVersion, #CurrentList];
}
public boolean unpack(container packedClass)
{
    Integer version = conPeek(packedClass,1);

    switch (version)
    {
        case #CurrentVersion:
            [version,#CurrentList] = packedClass;
            break;
        default:
            return false;
    }
    return true;
}
```

The overridden *run* method runs the operation. The following example prints the input parameters to the Infolog. It also prints the tier that the operation is running on and the run time that is used for execution.

```
public void run()
{
    if (xSession::isCLRSession())
    {
```

```
        info('Running in a CLR session.');
    }
    else
    {
        info('Running in an interpreter session.');
        if (isRunningOnServer())
        {
            info('Running on the AOS.');
        }
        else
        {
            info('Running on the Client.');
        }
    }

    info(strFmt('SysOpSampleBasicRunbaseBatch: %1, %2', this.parmNumber(), this.parmText()));
}
```

The *description* method provides a static description for the operation. Override the *description* method as shown in the following example to use this description as the default value for the caption shown in batch mode and in the user interface:

```
public static ClassDescription description()
{
    return 'Basic RunBaseBatch Sample';
}
```

Override the *main* method that prompts the user for input, and then runs the operation or adds it to the batch queue, as shown in the following example:

```
public static void main(Args _args)
{
    SysOpSampleBasicRunbaseBatch operation;

    operation = new SysOpSampleBasicRunbaseBatch();
    if (operation.prompt())
    {
        operation.run();
    }
}
```

The overridden *parmNumber* and *parmText* methods are optional. It is a Microsoft Dynamics AX best practice to expose operation parameters with the property pattern for better testability and for access to class member variables outside the class. Override these methods as shown in the following example:

```
public int parmNumber(int _number = number)
{
    number = _number;

    return number;
}
public str parmText(str _text = text)
{
```

```
    text = _text;

    return text;
}
```

The *main* method for the *RunBaseBatch* sample prompts the user for input for the operation when the *operation.prompt* method is called. If the prompt returns *true*, *main* calls the *operation.run* method directly. If the prompt returns *false*, it indicates that the user either canceled the operation or scheduled it to run as a batch.

To run the sample interactively, run the *main* method by clicking Go in the Code Editor window, as shown in Figure 14-2.

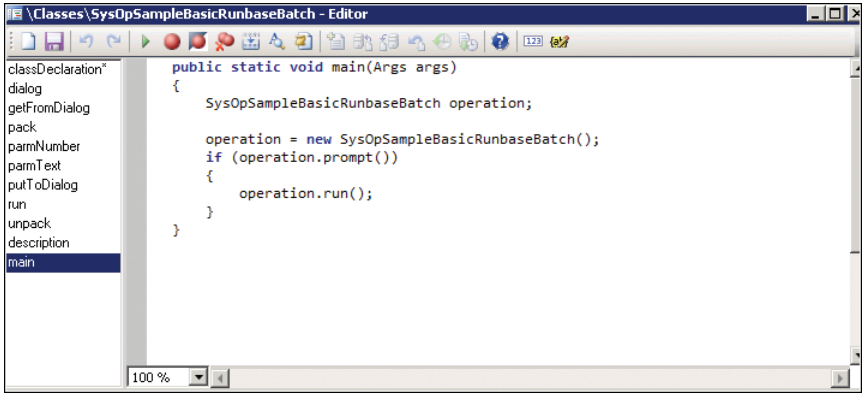


FIGURE 14-2 Code Editor window for *SysOpSampleBasicRunbaseBatch*.

On the General tab of the sample user interface, enter information in the Text Property and Number Property fields, as shown in Figure 14-3.

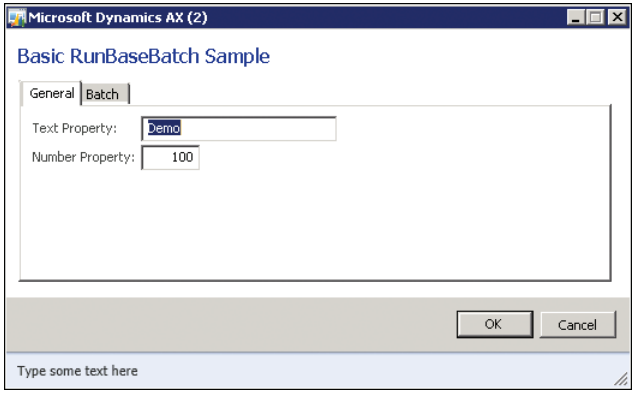


FIGURE 14-3 The General tab of the *SysOpSampleBasicRunbaseBatch* user interface.

On the Batch tab, ensure that the Batch Processing check box is cleared, as shown in Figure 14-4.

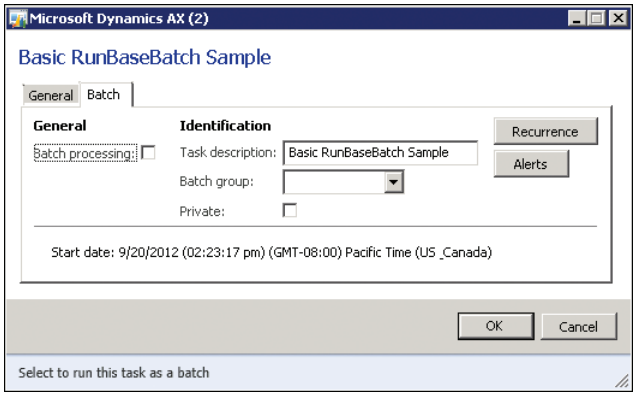


FIGURE 14-4 The Batch tab of the *SysOpSampleBasicRunbaseBatch* user interface.

Click OK to run the operation and print the output to the Infolog.

View the Infolog messages as shown in Figure 14-5. They show that the operation ran on the server because the sample *SysOpSampleBasicRunbaseBatch* class has the *RunOn* property set to *Server*. The operation ran by means of the X++ interpreter, which is the default for X++ code.

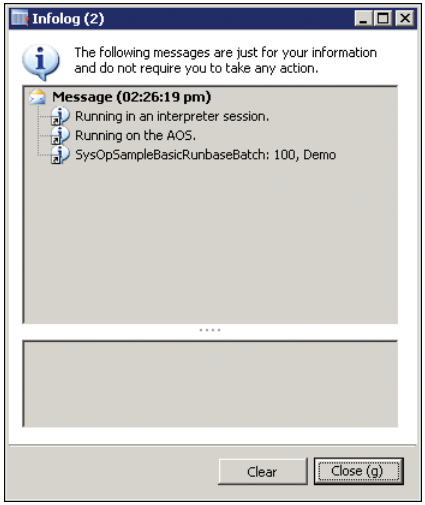


FIGURE 14-5 The Infolog window for *SysOpSampleBasicRunbaseBatch* output.

To run the sample in batch mode, rerun the operation by clicking Go in the Code Editor window and enter data for the *Text Property* and the *Number Property* on the General tab of the sample user interface.

Next, select the Batch Processing check box on the Batch tab to run the operation on the batch server. When the Batch Processing check box is selected, the Infolog message in Figure 14-6 appears, indicating that the operation has been added to the batch queue.

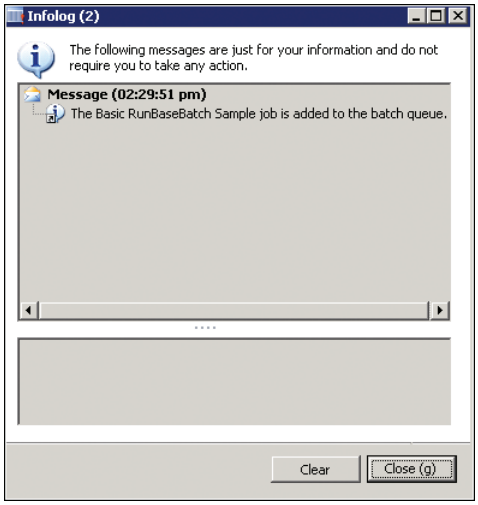


FIGURE 14-6 The Infolog window showing a job added to the batch queue.

The operation may take up to a minute to get scheduled. After waiting for about a minute, open the BatchJob form from the *Forms* node in the Application Object Tree (AOT), as shown in Figure 14-7.

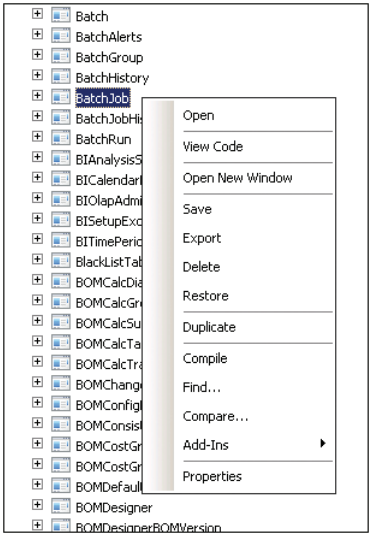


FIGURE 14-7 The *Forms* node in AOT.

The Job Description form opens, as shown in Figure 14-8.

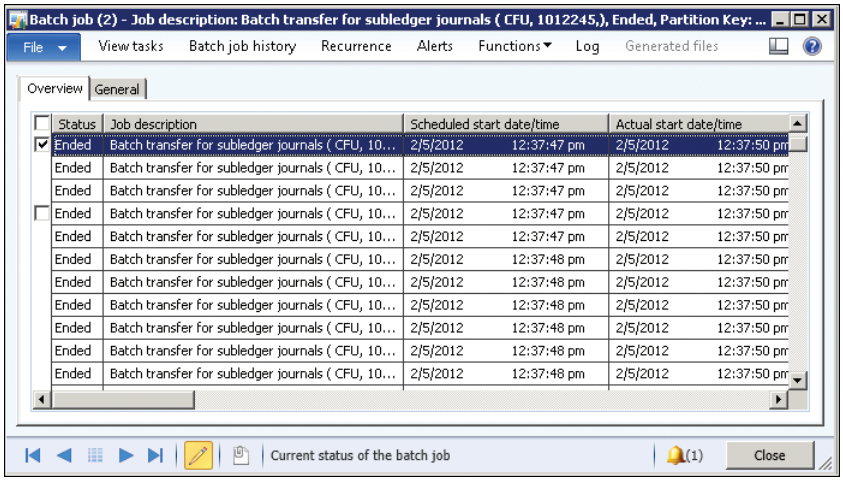


FIGURE 14-8 The Job Description form showing the status of batch jobs.

Press the F5 key to update the form. Press the F5 key repeatedly until the job entry shows that the job has ended. Sorting by the Scheduled Start Date/Time column may help you find the operation if there are many job entries in the grid.

To view the log, select the operation and then click Log on the toolbar.

The Infolog in Figure 14-9 illustrates messages indicating that the operation ran in a common language runtime (CLR) session, which is the batch server execution environment.

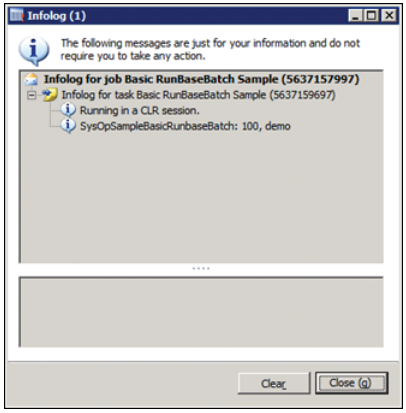


FIGURE 14-9 Messages for the *SysOpSampleBasicRunBaseBatch* sample.

SysOperation example: *SysOpSampleBasicController*

As mentioned earlier, the SysOperation framework provides the same capabilities as the RunBase framework but also includes base implementations for common overrides. The SysOperation framework handles basic user interface creation, parameter serialization, and routing to the CLR execution environment.

The SysOperation sample contains two classes: a controller class named *SysOpSampleBasicController* and a data contract class named *SysOpSampleBasicDataContract*. Table 14-2 describes the overridden methods that are necessary to match the functionality demonstrated in the RunBase sample in the previous section. Notice that you do not have to override the *dialog, getFromDialog, putToDialog, pack, unpack*, and *run* methods in the *SysOpSampleBasicController* class, because the SysOperation framework provides the base functionality for these methods. Example code later in this section illustrates how to use these methods.

TABLE 14-2 Method overrides for *SysOpSampleBasicController* and *SysOpSampleBasicDataContract* classes.

Method	Description
SysOpSampleBasicController class	
classDeclaration	Derives from the framework base class.
new	Identifies the class and method for the operation.
showTextInInfolog	Prints the input parameters, the tier that the operation runs on, and the run time used for execution to the Infolog.
caption	Provides a description for the operation.
main	Runs the operation.
SysOpSampleBasicDataContract class	
classDeclaration	The data contract attribute is used by the base framework to reflect on the operation.
parmNumber	The data member attribute identifies this property method as part of the data contract. The label, help text, and display order attributes provide hints for user interface creation.
parmText	See the description for *parmNumber*.

Important Normally, the *SysOpSampleBasicController* class would derive from *SysOperationServiceController*, which provides all of the base functionality for building operations; however, the Microsoft Dynamics AX 2012 version of the class contains a few known issues and these will be addressed in a future service pack. To work around the issues, a new common class, *SysOpSampleBaseController,* is available. For more information, see the white paper "Introduction to the SysOperation Framework," at *http://go.microsoft.com/fwlink/?LinkId=246316*.

Table 14-3 describes the issues and illustrates the solutions provided by the *SysOpSampleBaseController* class.

TABLE 14-3 Issues and workarounds for *SysOperationServiceController*.

Issue	Code in *SysOpSampleBaseController*

The controller should not be unpacked from the SysLastValue table when running as a batch.	`protected void loadFromSysLastValue()`
	`{`
	`  if (!dataContractsInitialized)`
	`  {`
	`  // This is a bug in the`
	`  // SysOperationController class`
	`  // never load from syslastvalue table`
	`  // when executing in batch`
	`  // it is never a valid scenario`
	`  // if (!this.isInBatch())`
	`  {`
	`  super();`
	`  }`
	
	`  dataContractsInitialized = true;`
	`  }`
	`}`

The default value for the *parmRegisterCallbackForReliableAsyncCall* property should be *false* to prevent unnecessary polling of the batch server.	`public void new()`
	`{`
	`  super();`
	
	`  // defaulting parameters common to all`
	`  // scenarios`
	
	`  // If using reliable async mechanism do not`
	`  // wait for the batch to`
	`  // complete. This is better done at the`
	`  // application level since`
	`  // the batch completion state transition is`
	`  // not predictable`
	`  // this.parmRegisterCallbackForReliableAsyncCa`
	`  // ll(false);`
	
	`  … code removed for clarity …`
	`}`

The default value for the *parmExecutionMode* property should be *Synchronous* to prevent issues when creating run-time tasks.	`public void new()`
	`{`
	`  … code removed for clarity …`
	
	`  // default for controllers in these samples`
	`  // is synchronous execution`
	`  // batch execution will be explicitly`
	`  // specified. The default for`
	`  // SysOperationServiceController is`
	`  // ReliableAsynchronous execution`
	
	`  this.parmExecutionMode(SysOperationExecution`
	`Mode::Synchronous);`
	
	`}`

The class declaration for *SysOpSampleBasicController* derives from the framework base class, *SysOpSampleBaseController*, which is provided with the sample code:

```
class SysOpSampleBasicController extends SysOpSampleBaseController
{
}
```

The *new* method for *SysOpSampleBasicController* identifies the class and method for the operation. In the following example, the *new* method points to a method on the controller class. However, it can point to any class method. The framework reflects on this class and method to provide the user interface and parameter serialization.

```
void new()
{
    super();

    this.parmClassName(
        classStr(SysOpSampleBasicController));
    this.parmMethodName(
        methodStr(SysOpSampleBasicController,
            showTextInInfolog));

    this.parmDialogCaption(
        'Basic SysOperation Sample');
}
```

In the following example, the *showTextInInfolog* method prints the input parameters, the tier where the operation is running, and the run time to the Infolog window:

```
public void showTextInInfolog(SysOpSampleBasicDataContract data)
{
    if (xSession::isCLRSession())
    {
        info('Running in a CLR session.');
    }
    else
    {
        info('Running in an interpreter session.');
        if (isRunningOnServer())
        {
            info('Running on the AOS.');
        }
        else
        {
            info('Running on the Client.');
        }
    }

    info(strFmt('SysOpSampleBasicController: %1, %2', data.parmNumber(), data.parmText()));
}
```

The *caption* method provides a description for the operation. This description is used as the default value for the caption shown in batch mode and the operation user interface.

```
public ClassDescription caption()
{
    return 'Basic SysOperation Sample';
}
```

The *main* method prompts the user for input and then runs the operation or adds it to the batch queue:

```
public static void main(Args args)
{
    SysOpSampleBasicController operation;

    operation = new SysOpSampleBasicController();
    operation.startOperation();
}
```

The three methods that you override in the *SysOpSampleBasicDataContract* class are shown in the following example. The framework uses the data contract attribute to reflect on the operation in the *class declaration*. The *parmNumber* and *parmText* methods use the data member attribute to identify these property methods as part of the data contract. The label, help text, and display order attributes provide hints for creating the user interface.

```
[DataContractAttribute]
class SysOpSampleBasicDataContract
{
    str text;
    int number;
}
[DataMemberAttribute,
SysOperationLabelAttribute('Number Property'),
SysOperationHelpTextAttribute('Type some number >= 0'),
SysOperationDisplayOrderAttribute('2')]
public int parmNumber(int _number = number)
{
    number = _number;

    return number;
}
[DataMemberAttribute,
SysOperationLabelAttribute('Text Property'),
SysOperationHelpTextAttribute('Type some text'),
SysOperationDisplayOrderAttribute('1')]
public Description255 parmText(str _text = text)
{
    text = _text;

    return text;
}
```

As in the RunBase sample, click Go on the Code Editor toolbar in the *main* method of the *SysOpSampleBasicController* class to run the SysOperation sample operation, as shown in Figure 14-10.

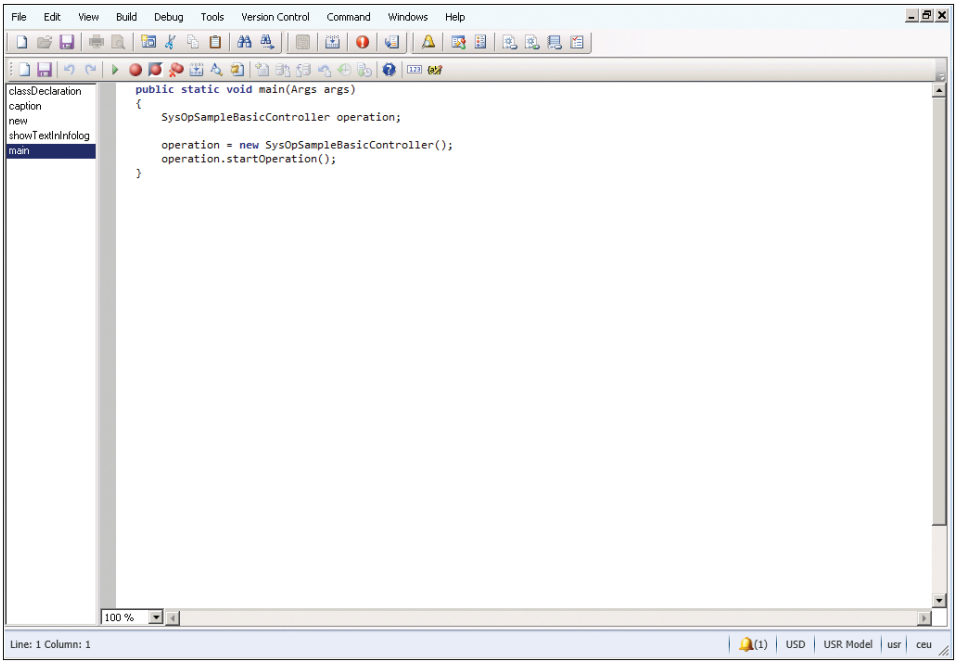


FIGURE 14-10 Running the SysOperation sample.

The *main* class calls *operation.startOperation*, which handles running the operation synchronously or adding it to the batch queue. The *startOperation* method invokes the user interface for the operation, and then calls run.

To run the operation interactively, enter information on the General tab of the operation user interface, as shown in Figure 14-11. The user interface created by the SysOperation framework is similar to the one created in the RunBase sample.

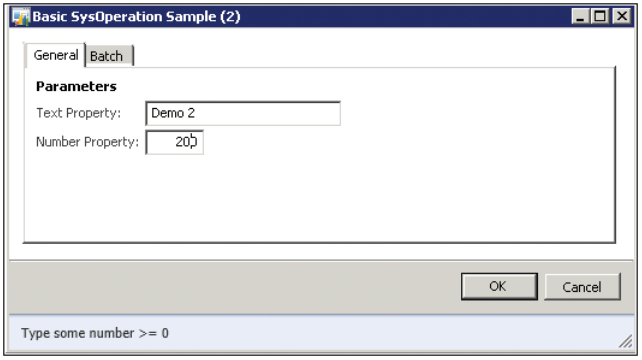


FIGURE 14-11 The General tab for the SysOperation framework example.

On the Batch tab, ensure that the Batch Processing check box is cleared, as shown in Figure 14-12.

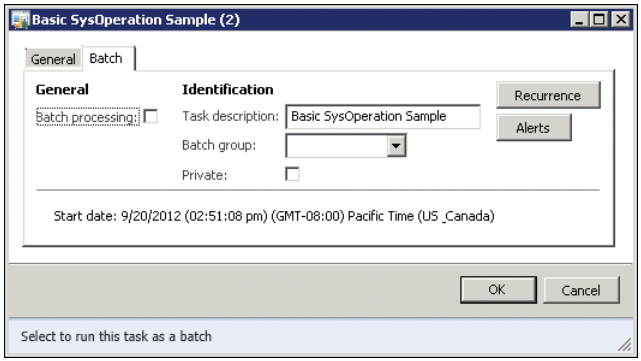


FIGURE 14-12 The Batch tab for the SysOperation framework example.

Click OK to run the operation and print the output to the Infolog window, as shown in Figure 14-13.

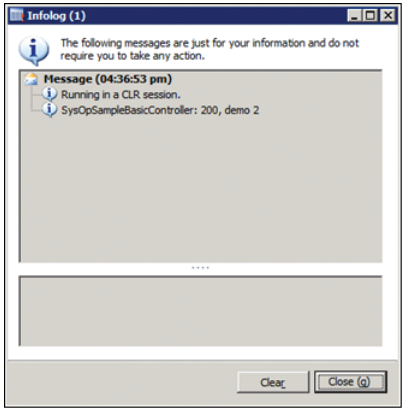


FIGURE 14-13 Infolog output for the SysOperation example.

The Infolog messages show that, unlike in the RunBase sample, the operation ran in a CLR session on the server.

If you repeat the previous steps but select the Batch Processing check box on the Batch tab, the operation runs on the batch server, just as in the RunBase sample.

The operation may take up to a minute to get scheduled. After waiting for about a minute, open the Batch Job form from the AOT.

Repeatedly update the form by pressing the F5 key until the job entry shows that the job has ended. Sorting by the Scheduled Start Date/Time column may help you find the operation if there are many jobs entries in the grid. After you find the correct job, select it, and then click Log on the toolbar to open an Infolog window.

Figure 14-14 shows that the operation ran in a CLR session, which is the batch server execution environment.

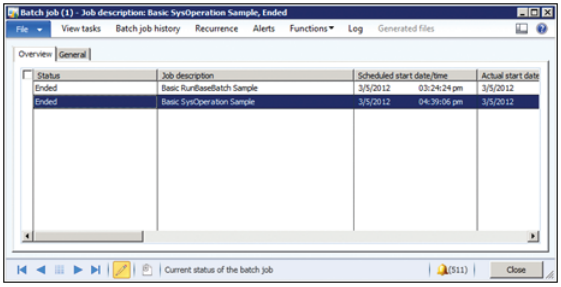


FIGURE 14-14 The Infolog window for the SysOperation example.

The RunBase framework

You can use the RunBase framework throughout Microsoft Dynamics AX whenever you must execute a business transaction job. Extending the RunBase framework lets you implement business operations that don't have default support in Microsoft Dynamics AX. The RunBase framework supplies many features, including dialog boxes, query windows, validation-before-execution windows, the progress bar, client/server optimization, pack-unpack with versioning, and optional scheduled batch execution at a given date and time.

Note Because the RunBase framework has largely been replaced by the SysOperation framework, the following sections are intended to help you understand existing functionality that uses the RunBase framework.

Inheritance in the RunBase framework

Classes that use the RunBase framework must inherit from either the *RunBase* class or the *RunBaseBatch* class. If the class extends *RunBaseBatch*, it can be enabled for scheduled execution in batch mode.

In a good inheritance model, each class has a public construction mechanism unless the class is abstract. If the class doesn't have to be initialized, use a static construct method. Because X++ doesn't support method name overloading, you should use a static *new* method if the class must be initialized further upon instantiation. For more information about constructors, see the section "Constructor encapsulation" in Chapter 4, "The X++ programming language."

Static *new* methods have the following characteristics:

- They are public.
- Their names are prefixed with *new*.
- They are named logically or with the arguments that they take. Examples include *newInventTrans* and *newInventMovement*.
- They usually take nondefault parameters only.
- They always return a valid object of the class type, instantiated and initialized, or throw an error.

Note A class can have several *new* methods with different parameter profiles. The *NumberSeq* class is an example of a class with multiple *new* methods.

The default constructor (the *new* method) should be protected to force users of the class to instantiate and initialize it with the static construct or *new* method. If *new* has some extra initialization logic that is always executed, you should place it in a separate *init* method.

Tip To make writing customizations easier, a best practice is to add construction functionality for new subclasses (in higher layers) without mixing code with the *construct* method in the original layer.

Property method pattern

To allow other business operations to run your new business operation, you might want to run it without presenting any dialog boxes to the user. If you decide not to use dialog boxes, you need an alternative to set the values of the necessary member variables of your business operation class.

In Microsoft Dynamics AX classes, member variables are always protected. In other words, they can't be accessed outside the class; they can be accessed only from within objects of the class or its subclasses. To access member variables from outside the class, you must write *accessor* methods. The *accessor* methods can get, set, or both get and set member variable values. All *accessor* methods start with *parm*. In Microsoft Dynamics AX, *accessor* methods are frequently referred to as *parm* methods.

Tip A Microsoft Dynamics AX best practice is not to use separate get and set *accessor* methods. The *accessor* methods are combined into a single *accessor* method, handling both get and set, in a pattern called the *property method pattern*. *Accessor* methods should have the same name as the member variable that they access, prefixed with *parm*.

The following is an example of how a method implementing the property method pattern could look:

```
public NoYesId parmCreateServiceOrders(NoYesId _createServiceOrders =
createServiceOrders)
{
    createServiceOrders = _createServiceOrders;

    return createServiceOrders;
}
```

If you want the method to work only as a *get* method, change it to something such as this:

```
public NoYesId parmCreateServiceOrders()
{
    return createServiceOrders;
}
```

And if you want the method to work only as a *set* method, change it to this:

```
public void parmCreateServiceOrders(NoYesId _createServiceOrders =
createServiceOrders)
{
    createServiceOrders = _createServiceOrders;
}
```

When member variables contain huge amounts of data (such as large containers or memo fields), the technique in the following example is recommended. This technique determines whether the parameter is changed. The disadvantage of using this technique in all cases is the overhead of an additional method call.

```
public container parmCode(container _code = conNull())
{
    if (!prmIsDefault(_code)
    {
        code = _code;
    }

    return code;
}
```

Pack-unpack pattern

When you want to save the state of an object with the option to reinstantiate the same object later, you must use the pack-unpack pattern. The RunBase framework requires that you implement this pattern to switch the class between client and server (for client/server optimization) and to present the user with a dialog box that states the choices made the last time the class executed. If your class extends the *RunBaseBatch* class, you also need to use the pack-unpack pattern for scheduled execution in batch mode.

The pattern consists of a *pack* method and an *unpack* method. These methods are used by the SysLastValue framework, which stores and retrieves user settings or usage data values that persist between processes.

Note A reinstantiated object is not the same object as the saved object. It is a copy of the object with the same values as the packed and unpacked member variables.

The *pack* method must be able to read the state of the object and return it in a container. Reading the state of the object involves reading the values of the variables needed to pack and unpack the object. Variables used at execution time that are declared as member variables don't have to be included in the *pack* method. The first entry in the container must be a version number that identifies the version of the saved structure. The following code is an example of the *pack* method:

```
container pack()
{
    return [#CurrentVersion, #CurrentList];
}
```

Macros must be defined in the class declaration. *CurrentList* is a macro defined in the *ClassDeclaration* holding a list of the member variables to pack. If the variables in the *CurrentList* macro are changed, the version number should also be changed to allow safe and versioned unpacking. The *unpack* method can support unpacking previous versions of the class, as shown in the following example:

```
class InventCostClosing extends RunBaseBatch
{
    #define.maxCommitCount(25)

    // Parameters

    TransDate                   transDate;
    InventAdjustmentSpec        specification;
    NoYes                       prodJournal;
    NoYes                       updateLedger;
    NoYes                       cancelRecalculation;
    NoYes                       runRecalculation;
    FreeTxt                     freeTxt;
    Integer                     maxIterations;
    CostAmount                  minTransferValue;
    InventAdjustmentType        adjustmentType;
    boolean                     collapseGroups;
    ...

    #DEFINE.CurrentVersion(4)
    #LOCALMACRO.CurrentList
        TransDate,
        Specification,
        ProdJournal,
        UpdateLedger,
```

```
        FreeTxt,
        MaxIterations,
        MinTransferValue,
        adjustmentType,
        cancelRecalculation,
        runRecalculation,
        collapseGroups
    #ENDMACRO

}
public boolean unpack(container packedClass)
{
    #LOCALMACRO.Version1List
        TransDate,
        Specification,
        ProdJournal,
        UpdateLedger,
        FreeTxt,
        MaxIterations,
        MinTransferValue,
        adjustmentType,
        del_minSettlePct,
        del_minSettleValue
    #ENDMACRO

    #LOCALMACRO.Version2List
        TransDate,
        Specification,
        ProdJournal,
        UpdateLedger,
        FreeTxt,
        MaxIterations,
        MinTransferValue,
        adjustmentType,
        del_minSettlePct,
        del_minSettleValue,
        cancelRecalculation,
        runRecalculation,
        collapseGroups
    #ENDMACRO

    Percent    del_minSettlePct;
    CostAmount del_minSettleValue;

    boolean         _ret;
    Integer         _version    = conpeek(packedClass,1);

    switch (_version)
    {
        case #CurrentVersion:
            [_version, #CurrentList] = packedClass;
            _ret = true;
            break;
```

```
        case 3:
            // List has not changed, just the prodJournal must now always be updated
            [_version, #CurrentList] = packedClass;
            prodJournal              = NoYes::Yes;
            updateLedger             = NoYes::Yes;
            _ret = true;
            break;

        case 2:
            [_version, #Version2List] = packedClass;
            prodJournal              = NoYes::Yes;
            updateLedger             = NoYes::Yes;
            _ret = true;
            break;

        case 1:
            [_version, #Version1List] = packedClass;
            cancelRecalculation       = NoYes::Yes;
            runRecalculation          = NoYes::No;
            _ret = true;
            break;

        default:
            _ret = false;
    }
    return _ret;
}
```

If any member variable isn't packable, the class can't be packed and reinstantiated to the same state. If any of the members are other classes, records, cursors, or temporary tables, they must also be made packable. Other classes that don't extend *RunBase* can implement the *pack* and *unpack* methods by implementing the SysPackable interface.

When the object is reinstantiated, it must be possible to call the *unpack* method, which reads the saved state and reapplies the values of the member variables. The *unpack* method can reapply the correct set of member variables according to the saved version number, as shown in the following example:

```
public boolean unpack(container _packedClass)
{
    Version    version = conpeek(_packedClass, 1);

    switch (version)
    {
        case #CurrentVersion:
            [version, #CurrentList] = _packedClass;
            break;

        default:
            return false;
    }
    return true;
}
```

The *unpack* method returns a Boolean value that indicates whether the initialization succeeded.

As mentioned earlier in this section, the *pack* and *unpack* methods have three responsibilities:

- Switching a *RunBase*-derived class between client and server
- Presenting the user with final choices made when the class was last executed
- Scheduling the execution of the class in batch mode

In some scenarios, it is useful to execute specific logic depending on the context in which the *pack* or *unpack* method is called. You can use the *isSwappingPrompt* method on *RunBase* to detect whether the *pack* or *unpack* method is called in the context of switching between client and server. The *isSwappingPrompt* method returns *true* when called in this context. You can use the *isInBatch* method on *RunBaseBatch* to detect whether the *unpack* method is called in the context of executing the class in batch mode.

Client/server considerations

Typically, you want to execute business operation jobs on the server tier because these jobs almost always involve several database transactions. However, you want the user dialog box to be executed on the client tier to minimize client/server calls from the server tier. Fortunately, both the SysOperation and the *RunBase* framework can help you run the dialog box on the client and the business operation on the server.

To run the business operation job on the server and push the dialog box to the client, you should be aware of two settings. On the menu item that calls the job, set the *RunOn* property to *Server*; on the class, set the *RunOn* property to *Called From*. For more information about these properties, see the section "Write tier-aware code, " in Chapter 13, "Performance."

When the job is initiated, it starts on the server, and the RunBase framework packs the internal member variables and creates a new instance on the client, which then unpacks the internal member variables and runs the dialog box. When the user clicks OK in the dialog box, RunBase packs the internal member variables of the client instance and unpacks them again in the server instance.

The extension framework

The extension framework is an extensibility pattern that reduces or eliminates the coupling between application components and their extensions. Many of the application foundation frameworks in Microsoft Dynamics AX are written by using the extension framework.

The extension framework uses the class attribute framework and the class factory framework to decouple base and derived classes in two steps.

Create an extension

First, create a class attribute method by extending the *SysAttribute* class.

An example can be found in the Product Information Management module in the *PCAdaptorExtensionAttribute* class:

```
class PCAdaptorExtensionAttribute extends SysAttribute
{
  PCName modelName;

  public void new(PCName _modelName)
  {
    super();
    if (_modelName == '')
    {
      throw error(Error::missingParameter(this));
    }
    modelName = _modelName;
  }

  public PCName parmModelName(PCName _modelName = modelName)
  {
    modelName = _modelName;
    return modelName;
  }
}
```

Next, use the *PCAdaptorExtensionAttribute* class attribute to add metadata to a derived class.

The following example code extends the *PCAdaptor* class to create a *MyPCAdaptor* object instead of a *PCAdaptor* object when you process a product configuration that is created from a product configuration model named *Computers*:

```
[PCAdaptorExtensionAttribute('Computers')]
class MyPCAdaptor extends PCAdaptor
{
  protected void new()
  {
    super();
  }
}
```

You can test this extension by performing the following steps:

1. Press Ctrl+Shift+W to open the Microsoft Dynamics AX Development Workspace.
2. Add the *MyPCAdaptor* class to the AOT and then compile the class.
3. Add a breakpoint in the *PCAdaptorFactory.getAdaptorFromModelName* method.
4. In the Microsoft Dynamics AX Windows client, click Product Information Management > Common > Product Configuration Models.

5. On the Action Pane, in the New group, click Product Configuration Model. The New Product Configuration Model dialog box opens.

6. In the *Name* field, enter **Computers**.

7. Enter a name in the *Root Component Section Name* field, and then click OK. The Constraint-based Product Configuration Model details form opens.

8. In the Attributes section of the form, add an attribute for the root component. For example, size and color are common attributes.

9. On the Action Pane, in the Run group, click Test.

10. Select a value for the attribute.

The Microsoft Dynamics AX debugger launches at the breakpoint that you added in step 3.

As you step through the code, you can see how the *SysExtensionAppClassFactory* is used to create an instance of the *MyPCAdaptor* class:

```
adaptor = SysExtensionAppClassFactory::getClassFromSysAttribute(
    classStr(PCAdaptor), extensionAttribute);
```

The *getClassFromSysAttribute* method works by searching through the classes that are derived from the *PCAdaptor* class. It returns an instance when it finds a class that has a *PCAdaptorExtensionAttribute* that returns a product model name that matches the name of the product configuration model passed in. In this case, an instance is created for the product configuration model named *Computers*.

Your custom code benefits by using this extension model because the base and derived classes are decoupled and it takes less code to extend the capabilities of Microsoft Dynamics AX.

Extension example

The following end-to-end example shows how to write extensible classes and presents some sample extensions.

First, create a class derived from *SysAttribute* called *CalendarExtensionAttribute*, which can be used to mark a class as extensible:

```
public class CalendarExtensionAttribute extends SysAttribute
{
    str calendarType;
}

public void new(str _calendarType)
{
    super();
    if (_calendarType == '')
    {
```

```
        throw error(error::missingParameter(this));
    }
    calendarType = _calendarType;
}

public str parmCalendarType(str _calendarType = calendarType)
{
    calendarType = _calendarType;
    return calendarType;
}
```

Next, use the newly-created attribute class to add metadata to the extensible *Calendar* class and its derived classes:

```
[CalendarExtensionAttribute("Default")]
public class Calendar
{
}

public void new()
{
}

public void sayIt()
{
    info("All days are work days except for weekends!");
}
```

The following code illustrates two sample extensions, a *FinancialCalendar* and a *HolidayCalendar*. Both classes override the *sayIt* method:

```
[CalendarExtensionAttribute("Financial")]
public class FinancialCalendar extends Calendar
{
}

public void sayIt()
{
    super();
    info("Financial Statements are available on the last working day of June!");
}
[CalendarExtensionAttribute("Holiday")]
public class HolidayCalendar extends Calendar
{
}

public void sayIt()
{
    super();
    info( "Eight public holidays including New Year's Day!");
}
```

Finally, a custom factory class is created to generate the appropriate instance of the *Calendar* class. This custom factory class uses *SysExtensionAppClassFactory.getClassFromSysAttribute* method which searches through the derived classes of the *Calendar* class to match the parameters of their attribute metadata with the parameters in the call. The following code shows the *CalendarFactory* class that creates a calendar instance:

```
public class CalendarFactory
{
}

public static Calendar instance(str _calendarType)
{
    CalendarExtensionAttribute extensionAttribute =
        new CalendarExtensionAttribute(_calendarType);
    Calendar calendar =
        SysExtensionAppClassFactory::getClassFromSysAttribute(classStr(calendar),
extensionAttribute);

    if (calendar == null)
    {
        calendar = new Calendar();
    }

    return calendar;
}
```

The following code contains a job that shows possible calendar creation scenarios:

```
static void CreateCalendarsJob(Args _args)
{
    Calendar calendar = CalendarFactory::instance("Holiday");
    calendar.sayIt();
    calendar = CalendarFactory::instance("Financial");
    calendar.sayIt();
    calendar = CalendarFactory::instance("Default");
    calendar.sayIt();
}
```

Eventing

Eventing is another extensibility pattern that reduces or eliminates the coupling between application components and their extensions. While the extension framework is coarse-grained and suitable for plug-ins, eventing can be fine-grained. You can use it to augment or modify existing application behaviors effectively.

The following terms are related to events in X++:

- **Producer** The logic that contains the code that causes a change. It is an entity that emits events.
- **Consumer** The application code that represents an interest in being notified when a specific event occurs. It is an entity that receives events.

- **Event** A representation of a change having happened in the producer.
- **Event payload** The information that the event carries with it. When a person is hired, for example, the payload might include the employee's name and date of birth.
- **Delegate** The definition of the information that is passed from the producer to the consumer when an event takes place.

By using events, you can potentially lower the cost of creating and upgrading customizations. If you create code that is often customized by others, you can create events in places where customizations typically occur. Then, developers who customize the original functionality in another layer can subscribe to an event. When customized functionality is tied to an event, the underlying application code can be rewritten with little impact on the customization, so long as the same events are raised in the same sequence from one version to the next.

You can use events to support the following programming paradigms:

- **Observation** Events can detect exceptional behavior and generate alerts when such behavior occurs. For example, this type of event might be used in a regulation-compliance system. If more than a designated amount of money is transferred from one account to another, an event can be raised and event handlers can respond to the event appropriately. For example, the event handlers could reject the transaction and send an alert to the account manager.
- **Information dissemination** Events can deliver the right information to the right consumers at the right time. Information can be disseminated by publishing an event to anyone who wants to react to it. For example, the creation of a new worker in the system might be of interest to Human Resources employees who conduct new employee orientations.
- **Decoupling** Events produced by one part of the application can be consumed by a different part of the application. The producer does not have to be aware of the consumers, nor do the consumers need to know details about the producer. One producer's event can be acted upon by any number of consumers. Conversely, consumers can act upon any number of events from many different producers. For example, the creation of a new worker in the system might be consumed by the Project Management and Accounting module, if you want to include the worker in default project teams.

Microsoft Dynamics AX events are based on .NET eventing concepts. For more information, see "X++, C# Comparison: Event" at *http://msdn.microsoft.com/en-us/library/gg881685(v=ax.60).aspx*.

Delegates

In X++, you can add delegates as members of a class. The syntax for defining a delegate is the same as the syntax used for defining a method, with the following exceptions:

- The delegate keyword is used.
- No access modifiers can be used on a delegate declaration because all delegates are protected members.

- The return type must be void.
- The body must be empty; that is, it can contain neither declarations nor statements.
- A delegate can be declared only as a member of a class. A delegate cannot be a member of a table.

For example, a delegate for an event that is raised when a person is hired could be expressed like this:

```
delegate void hired(str personnelNumber, UtcDateTime startingDate)
{
   // Delegates do not have any code in the body
}
```

The parameters defined in the parameter profile can be any type allowed in X++. In particular, it is useful to pass an object instance and to have the handlers modify the state of the object. In this way, the publisher can solicit values from the subscribers.

Pre and *post* events

Pre and *post* events are predefined events that occur when methods are called. *Pre* event handlers are called before the designated method executes, and *post* event handlers are called after the method call has ended. You can think of these event handlers as augmenting an existing method with additional methods that are called before and after the designated method. The event handlers for these *pre* and *post* events are visible in the AOT as subnodes of the methods to which they apply.

The following pseudocode illustrates a method without event handlers:

```
void someMethod(int i)
{
   --body of the method--
}
```

The following example shows the method after event handlers are added:

```
void someMethod(int i)
{
   preHandler1(i);
   preHandler2(i);
   --body of the method–
   postHandler1(i);
   postHandler2(i);
}
```

Not having any event handlers for a particular method leaves the method intact. Therefore, no overhead is incurred for methods that do not have any *pre* or *post* handlers assigned to them.

If an exception is thrown in a *pre* event handler, neither the remaining event handlers nor the method itself is invoked. If a method that has any *pre* event or *post* event handlers throws an exception, the remaining *post* event handlers are not invoked. If an exception is thrown in a *post* event handler, the remaining event handlers are not called.

Each *pre* event handler can access the original values of the parameters and modify them as required. A *post* event handler can modify the return value of the method.

Event handlers

Event handlers are the methods that are called when the delegate is called, either directly through code (for coded events) or from the environment (for modeled events that are maintained in the AOT). The relationship between the delegate and the handlers can be maintained in the code or in the AOT.

To add an event handler declaratively in the AOT, you identify a static method to handle the event on the delegate, and then simply drag the method to the delegate node that represents the event to be handled. You can remove an event handler by using the Delete menu item that is available for any node in the AOT. You can use only static methods in this context.

To add a static event handler in code, you use a special X++ syntax, as shown in the following example:

```
void someMethod(int i)
{
   this.MyDelegate += eventhandler(Subscriber::MyStaticHandler);
}
```

The delegate name appears on the left side of the += operator. On the right side, you can see the keyword *eventhandler*, along with the qualified name of the handler to add. The compiler checks that the parameter profiles of the delegate and the handler match. The qualified name in the example uses two colon characters (*::*) to separate the type name and the delegate, which designates that the event handler is static.

To call a method on a particular object instance, use the syntax shown in the following example:

```
void someMethod(int i)
{
   EventSubscriber subscriber = new EventSubscriber();
   this.MyDelegate += eventhandler(subscriber.MyInstanceHandler);
}
```

You can remove the event handler from a delegate by using the -= operator instead of the += operator. An example of removing a static event handler is as follows:

```
void someMethod(int i)
{
   this.MyDelegate -= eventhandler(Subscriber::MyHandler);
}
```

Here are some things to keep in mind about events:

- The X++ compiler does not allow you to raise events from outside the class in which the delegate is defined.
- The run-time environment makes no guarantees about the order in which the event handlers are called.
- An event handler can be implemented either in managed code or in X++. You define managed code event handlers in a Microsoft Visual Studio project that you add to the AOT.

Eventing example

The following example illustrates the use of both coded and modeled events. The example also shows the ways in which arguments can be passed to event handlers.

The following code implements an array with an event indicating that an element has changed:

```
public class arrayWithChangedEvent extends Array
{
}

delegate void changedDelegate(int _index, anytype _value)
{
}

public anytype value(int _index, anytype _value = null)
{
    anytype paramValue = _value;
    anytype val = super(_index, _value);
    boolean newValue = (paramValue == val);
    if (newValue)
        this.changedDelegate(_index, _value);

    return val;
}
```

The following dynamic event handler is added at run time:

```
public class arrayChangedEventListener
{
    arrayWithChangedEvent arrayWithEvent;
}

public void new(ArrayWithChangedEvent _arrayWithEvent)
{
    arrayWithEvent = _arrayWithEvent;

    // Register the event handler with the delegate
    arrayWithEvent.ChangedDelegate += eventhandler(this.ListenToArrayChanges);
}
```

```
public void listenToArrayChanges(int _index, anytype _value)
{
    info(strFmt("Array changed at: %1 - with value: %2", _index, _value));
}

public void detach()
{
    // Detach event handler from delegate
    arrayWithEvent.changedDelegate -= eventhandler(this.listenToArrayChanges);
}
```

The following example contains two static event handlers:

```
public static void ArrayPreHandler(XppPrePostArgs args)
{
    int indexer = args.getArg("_index");
    str strVal = "";
    if (args.existsArg("_value") && typeOf(args.getArg("_value")) == Types::String)
    {
        strVal = "Pre-" + args.getArg("_value"); // Mark the value as Pre- processed
        args.setArg("_value", strVal);
        // The changes to parameter values may be based on
        // state of the record or environment variables.
    }
}

public static void ArrayPostHandler(XppPrePostArgs args)
{
    anytype returnValue = args.getReturnValue();
    str strReturnValue = "";

    if (typeOf(returnValue) == Types::String)
    {
        strReturnValue = returnValue + "-Post"; // post- mark the return value
        args.setReturnValue(strReturnValue);
    }
}
```

To exercise the eventing example, add the *pre* and *post* event handlers to the *value* method of the *ArrayWithChangedEvent* class in the AOT, and then run the following job:

```
static void EventingJob(Args _args)
{
    // Create a new array
    ArrayWithChangedEvent arrayWithEvent = new ArrayWithChangedEvent(Types::String);

    // Create listener for the array
    ArrayChangedEventListener listener = new ArrayChangedEventListener(arrayWithEvent);

    // Test by adding items to the array
    info(arrayWithEvent.value(1, "Blue"));
    info(arrayWithEvent.value(2, "Cerulean"));
    info(arrayWithEvent.value(3, "Green"));
```

```
    // Detach listener from array
    listener.Detach();

    // The following additions should not invoke the listener,
    // except when any pre and post events exist
    info(arrayWithEvent.value(4, "Orange"));
    info(arrayWithEvent.value(5, "Pink"));
    info(arrayWithEvent.value(6, "Yellow"));
}
```

CHAPTER 15

Testing

In this chapter

Introduction 527
New unit testing features in Microsoft Dynamics AX 2012 . . . 527
Microsoft Visual Studio 2010 test tools 533
Putting everything together 540

Introduction

Ensuring a quality user experience with an enterprise resource planning (ERP) product like Microsoft Dynamics AX 2012 can be challenging. Out-of-the-box from Microsoft, the product is broad, deep, and complex. A typical customer deployment adds one or more independent software vendor (ISV) products to better tailor the product for specific purposes. Finally, a customer or partner customizes Microsoft Dynamics AX to further align it with company processes and policies.

The end user doesn't see or, for that matter, care about the complexities of the product or who delivered what functionality. The end user wants a product that enables her to perform her daily tasks efficiently, effectively, and with a good user experience. The entire ecosystem—Microsoft, ISV developers, Microsoft Dynamics AX partners, and customer IT developers—is part of a critical quality-assurance link that end users depend on when using Microsoft Dynamics AX 2012 to accomplish their goals.

This chapter focuses on new features and tools that facilitate improved testing of Microsoft Dynamics AX 2012 ISV solutions and customizations. This release includes several new capabilities that collectively take a major step forward in the ability to effectively test Microsoft Dynamics AX 2012 solutions.

New unit testing features in Microsoft Dynamics AX 2012

The SysTest framework for unit testing has been part of the Microsoft Dynamics AX MorphX environment for several releases. The framework can be very useful for traditional unit testing of X++ classes and methods. The framework can also be used for integration testing of business logic that spans multiple classes.

Past editions of the "Inside Microsoft Dynamics AX" series and the current MSDN documentation on the SysTest framework do a very good job of explaining the framework basics. By making use of the new attribute capabilities in the X++ language, the SysTest framework in Microsoft Dynamics AX 2012 has new capabilities that you can use to develop and execute your tests more flexibly. This section focuses on these features.

Use predefined test attributes

X++ attributes are a new feature in Microsoft Dynamics AX 2012 and are described in Chapter 4, "The X++ programming language." This section describes how you can use the predefined test attributes in the SysTest framework to improve development and execution flexibility.

Five SysTest attributes are provided as part of the framework. These attributes are described in Table 15-1.

TABLE 15-1 Predefined SysTest attributes.

Test Attribute	Description	Where to Apply the Attribute
SysTestMethodAttribute	Indicates that a method is a unit test.	Apply the attribute to a method.
SysTestCheckInTestAttribute	Indicates that the test is a check-in unit test. A check-in test is run when checking in code to a version control system to ensure a proper level of quality.	Apply the attribute to a method or a class.
SysTestNonCheckInTestAttribute	Indicates that the test is not a check-in test.	Apply the attribute to a method.
SysTestTargetAttribute(<name>, <type>)	Indicates the application object that is being tested by the test case; for example, class, table, or form. This attribute takes two parameters: ■ *Name* String value name of the code element to test. ■ *Type* The type of the code element to test.	Apply the attribute to a class.
SysTestInactiveTestAttribute	Indicates that the class or method is deactivated.	Apply the attribute to a method.

Naming conventions were used heavily in previous versions of the SysTest framework to indicate the intent of a class or a method. A class naming convention of *<TargetClass>Test* was used to specify the code class targeted for code coverage collection. (You could also override the *testsElementName* method on your test class as an alternative to this convention.) All methods intended to be test methods in a SysTestCase-derived class had to start with the string *test*.

By using *SysTestTargetAttribute* and *SysTestMethodAttribute,* you can be more explicit in your unit test code. The following code examples show you how to use these attributes.

```
[SysTestTargetAttribute(classStr(Triangles), UtilElementType::Class)]
public class TrianglesTest extends SysTestCase
{
}

[SysTestMethodAttribute]
public void testEQUILATERAL()
{
    Triangles triangle = new Triangles();
    this.assertEquals(TriangleType::EQUILATERAL, triangle.IsTriangle(10, 10, 10));
}
```

The new predefined attributes provide the capability to create filters so that you can execute specific tests shown in Figure 15-1. You can use *SysTestCheckInTestAttribute*, *SysTestNonCheckInTestAttribute*, and *SysTestInactiveTestAttribute* for this purpose. In the Parameters form, which you access from the unit test toolbar, you can select the filter you want to use when running tests. For information about how to create a filter, see the following section.

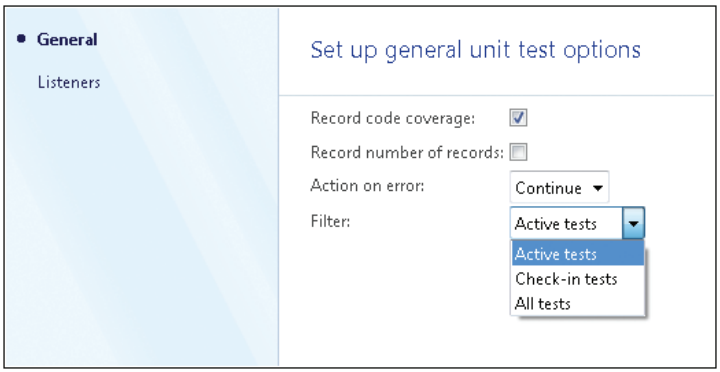


FIGURE 15-1 Filter list in the Parameters form.

Executing regression tests when code is checked in to version control is a great way to keep your code base at a high level of quality. But you don't always want to run all tests because the amount of time required for execution might become an issue. This is where the attributes *SysTestCheckInTestAttribute* and *SysTestNonCheckInTestAttribute* are useful. For more information about version control, see Chapter 2, "The MorphX development environment and tools."

Use the following steps to specify which tests are executed on check-in:

1. Attribute test methods with *SysTestCheckInTestAttribute* or *SysTestNonCheckInTestAttribute*. Note that the default option is for a test to *not* be a check-in test, so you have to specify the *SysTestCheckInTestAttribute* to opt in. The best approach is to be explicit with all tests, as shown in this example.

```
[SysTestMethodAttribute,
SysTestCheckInTestAttribute]
public void testEQUILATERAL()
{
    Triangles triangle = new Triangles();
    this.assertEquals(TriangleType::EQUILATERAL, triangle.IsTriangle(10, 10, 10));
}
```

2. Create a new test project and put all unit test classes with check-in tests into the project.
3. In the Settings dialog box for the test project that you created (right-click the name of the project, and then click Settings), specify Check-in Tests as the filter.
4. In System Settings for Version Control (on the Version Control menu, click System Settings), select your test project in the Test project list.

On your next check-in, the appropriate tests will execute, and the results will be displayed in an Infolog message.

Create test attributes and filters

The predefined test attributes described in the previous section are a good starting point for being more explicit in your test code and for organizing your tests. A well-organized strategy for using test projects can also be helpful. But there's a good chance that you will want to take test organization a step further for your development projects. Fortunately, you can extend the test attribute capabilities by creating your own attributes and filters.

As noted earlier in this chapter, the SysTest framework can be useful for both class-level unit testing and for integration tests on business logic that spans classes. However, it might be useful to be able to run just the integration tests in certain scenarios because they are more functionally oriented. For example, you might want to run only the integration tests when moving code from your test environment into preproduction.

This section demonstrates how you can create a new attribute and a corresponding filter to use on integration tests.

First, create the new attribute. This is quite straightforward because you only need to create a class that inherits from *SysTestFilterAttribute*.

```
class SysTestIntegrationTestAttribute extends SysTestFilterAttribute
{
}
```

You can now use this attribute on a new test method as follows:

```
[SysTestMethodAttribute,
SysTestIntegrationTestAttribute]
public void testIntegratedBusinessLogic()
{
    this.assertFalse(true);
}
```

While this attribute is informative to anyone reading the test code, it isn't useful for execution until you also enable it in the Filter drop-down list for test selection. To do this, you need to implement a *test strategy* for the *IntegrationTestAttribute*. The term "*strategy*" is used because the test strategy is implemented following the Strategy design pattern.

First, extend the *SysTestFilterStrategyType* enumeration with an additional element, as shown in Figure 15-2. Remember to set an appropriate label on the Properties sheet.

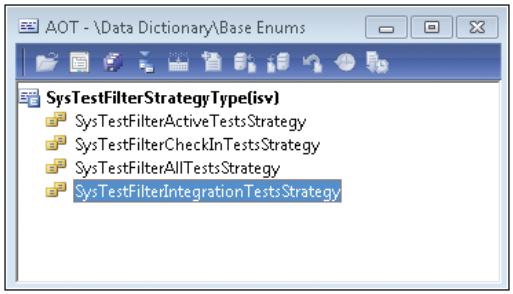


FIGURE 15-2 Extension to the *SysTestFilterStrategyType* enumeration.

Next, implement the strategy in a class with the same name as the enumeration element name. This class inherits from *SysTestFilterStrategy* and has a class declaration, as shown in the following example:

```
class SysTestFilterIntegrationTestsStrategy extends SysTestFilterStrategy
{
}
```

The most straightforward way to implement this strategy is to follow the pattern in one of the other *SysTestFilter<attribute>TestsStrategy* classes. You need to implement only two methods in this case.

The construct method returns a new instance of the class. You will use this method shortly.

```
public static SysTestFilterIntegrationTestsStrategy construct()
{
    return new SysTestFilterIntegrationTestsStrategy();
}
```

The work in this class is being done in the *isValid* method. This method determines if a test method should be included in the list of selected tests. For *SysTestFilterIntegrationTestsStrategy*, here is the implementation.

```
public boolean isValid(classId _classId, identifierName _method)
{
    SysDictMethod method;
    DictClass dictClass;

    method = this.getMethod(_classId, _method);
    if (method)
    {

        //
        // If the test method has the integration attribute, include it.
        //
        if (method.getAttribute(attributestr(SysTestIntegrationTestAttribute)))
        {
            return true;
        }
    }

    //
    // If the test class has the integration attribute, include it.
    //
    dictClass = new DictClass(_classId);
    if (dictClass.getAttribute(attributestr(SysTestIntegrationTestAttribute)))
    {
        return true;
    }
    return false;
}
```

Note Additional code is required to achieve the desired behavior of a *SysTestInactiveTestAttribute* that could also be used on the test method. This code was omitted to keep the example simple.

There is one last thing to do to enable the new integration test attribute. The *newType* method in the *SysTestFilterStrategy* class creates the appropriate type based on the selection in the Filter list and must have an additional case added to it, as shown in the following example:

```
public static SysTestFilterStrategy newType(SysTestFilterStrategyType _type)
{
    SysTestFilterStrategy strategy;

    switch (_type)
    {
             <snip – non essential code removed>
        // Create an integration test strategy
        case SysTestFilterStrategyType::SysTestFilterIntegrationTestsStrategy:
            strategy = SysTestFilterIntegrationTestsStrategy::construct();
            break;
```

```
            default:
                throw error(error::wrongUseOfFunction(funcname()));
        }

        strategy.parmFilterType(_type);
        return strategy;
    }
```

The Integration Tests option is now available in the Filter list (Figure 15-3) and, when selected, will run only those test methods attributed with *SysTestIntegrationTestAttribute*.

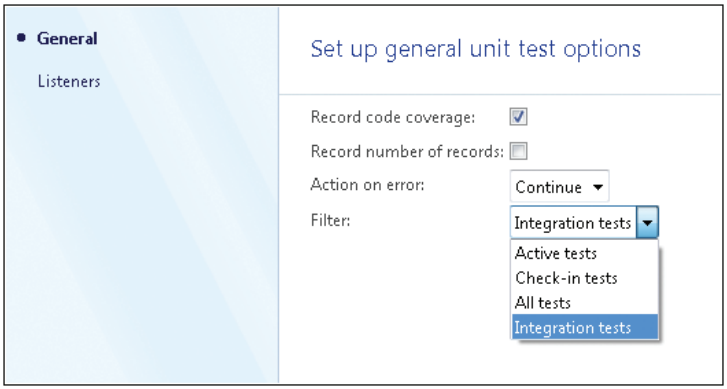


FIGURE 15-3 A filter list with a custom filter.

Microsoft Visual Studio 2010 test tools

While the SysTest unit testing capabilities are critical for developers who are testing Microsoft Dynamics AX, much of the testing is not done by developers. The functional testing, validating that the product meets the customer requirements, is typically done by someone with a title like functional consultant, business analyst, or possibly someone whose primary job involves using Microsoft Dynamics AX to accomplish daily tasks.

These functional testers have a number of things in common:

- They are experts in the product and the application of the product to solve business needs.
- They are not trained as software developers or software testers. Any non-trivial programming required for their test efforts is challenging.
- They are not trained as software testers, but they are typically quite good at testing. They have an inquisitive nature that gives them a knack for finding issues in the product.
- They would love to have automated test support for repetitive tasks, but they also believe that using an exploratory manual testing approach is the best way to validate the system and find critical issues.

Microsoft Visual Studio 2010 Ultimate and Visual Studio Test Professional contain Microsoft Test Manager, an application that was designed with these types of testers in mind. This package of testing tools is well suited for Microsoft Dynamics AX projects. Team Foundation Server, which is required for Microsoft Test Manager, brings several other quality-focused benefits to the table. It provides an application lifecycle management (ALM) solution for the development phase of the project by integrating requirements management, project management, source code control, bug tracking, build processes, and test tools together. For more information about ALM, see the white paper, "What Is Application Lifecycle Management?" at *http://www.microsoft.com/global/applicationplatform/en/us/RenderingAssets/Whitepapers/What%20is%20Application%20Lifecycle%20Management.pdf*.

This section focuses on best practices for applying Microsoft Test Manager to Microsoft Dynamics AX projects. For more information about how to use Microsoft Test Manager, see "Quick Start Guide for Manual Testing Using Microsoft Test Manager" on MSDN (*http://msdn.microsoft.com/en-us/library/dd380763.aspx*).

Use all aspects of the ALM solution

The quality plan for your project should not be focused on testing alone. On the contrary, many factors can have a bigger impact on the quality of the project than the amount of testing done. One key quality factor is *configuration management*. With the Visual Studio ALM solution, you can track all of the significant artifacts in your software development process. You can drive higher quality in your projects by adopting the ALM solution throughout your project.

Figure 15-4 describes an end-to-end feature development cycle involving Simon, a functional consultant, and Isaac, an IT developer. As you can see, the process involves tracking requirements, test cases, source code, builds, and bugs. Many of these items are tracked in Team Foundation Server (TFS). It also describes traceability between these artifacts. You could also incorporate work items into the process for improved project management.

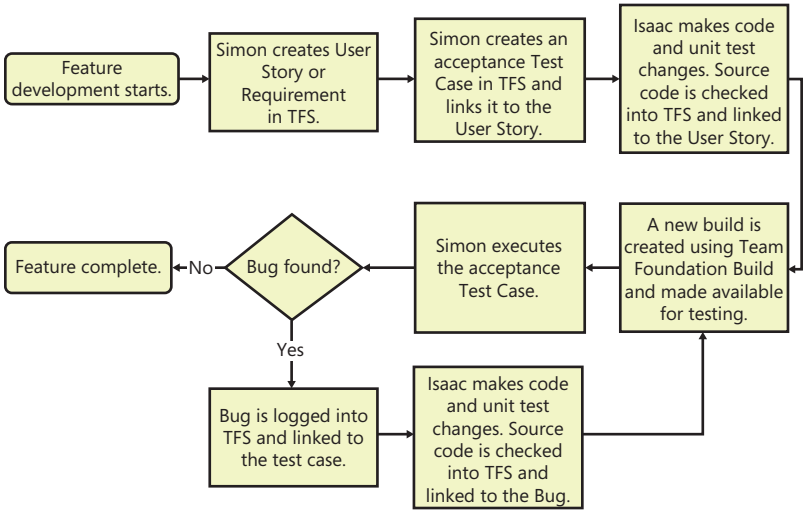


FIGURE 15-4 Feature development cycle.

Use an acceptance test driven development approach

In a rapidly changing environment like most Microsoft Dynamics AX projects, an agile development approach is often the best development methodology. One agile development practice that is particularly helpful in ensuring quality is acceptance test driven development (ATDD). ATDD involves defining the critical test cases, the acceptance test cases, ahead of the development effort as a requirement. (The term "*requirement*" is used generically here. The requirement can be a user story, a feature, or another artifact that describes functionality that is valuable to a customer.)

Acceptance test cases frequently exercise the primary flows for a requirement. Additional test cases are required to fully test the requirement. Acceptance test cases should be a collaborative effort between the developer, the tester, and the author of the requirement. A significant benefit of this collaboration is clarity because the individuals involved frequently have different, unstated versions of how they expect the requirement to be implemented.

While the requirement should be free of implementation details, the acceptance test cases must have some implementation details to make them useful—but not too many. Figure 15-5 shows a sample acceptance test case for a feature described in the previous section—executing unit tests when code is checked in. The test case specifies details such as the forms that are expected to be used, but it doesn't specify the exact field names.

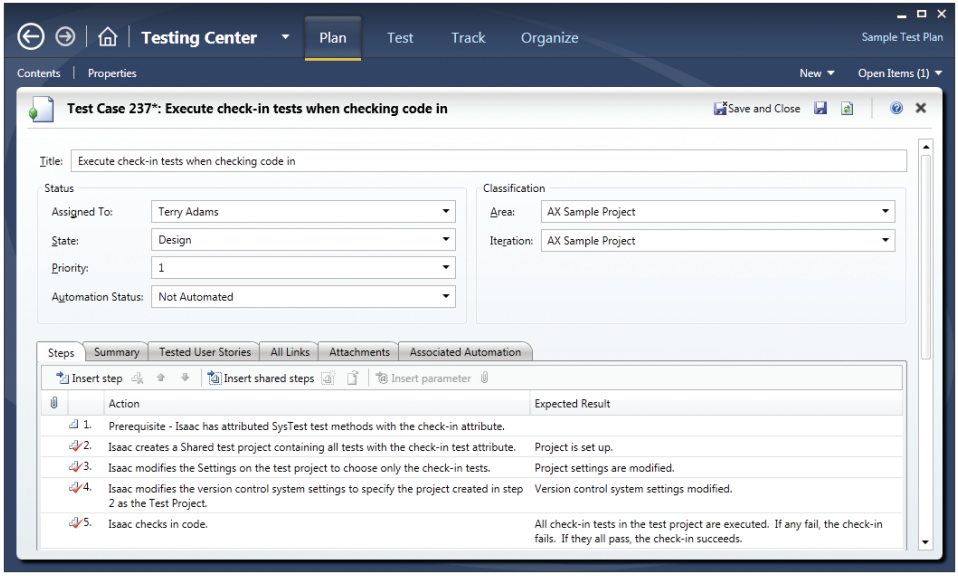


FIGURE 15-5 Acceptance test case.

After you create the test, link it to the requirement on the Tested User Stories tab. (In this example, a user story is the requirement.) The Add Link form will look like Figure 15-6 after linking.

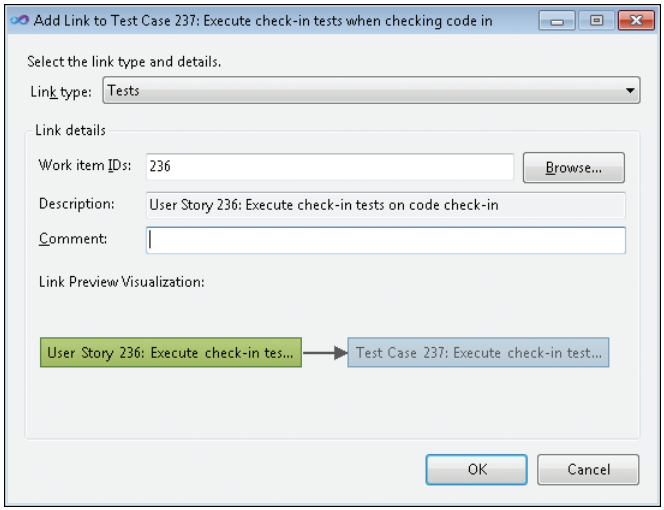


FIGURE 15-6 Requirement linked to test case.

By linking the test case to the requirement, you can use a nice feature in Microsoft Test Manager, the capability to build test plans based on requirement-based test suites. By specifying the requirement using the Add Requirements button in the Microsoft Test Manager Plan area, you pull in all test cases that are linked to the requirement.

Use shared steps

With shared steps, you can reuse the same steps in multiple test cases. Think of shared steps as subroutines for your manual test cases. Using this capability appropriately is a big step toward long-term maintainability of your test cases.

A prime opportunity for using shared steps is to get the application into a known state at the start of each test case. For Microsoft Dynamics AX tests, starting the application through the command line and using command-line options to initialize the application for testing is an excellent strategy.

Here's an example of how to start Microsoft Dynamics AX and run a job to initialize data. First, create an XML file, *fminitializedata.xml*, that you will reference in the command line and save it in a well-known location.

Note While this example uses the root folder of the C drive, a better approach would be to define an environment variable for the location.

```
<?xml version="1.0" ?>
<AxaptaAutoRun
    exitWhenDone="false"
    logFile="c:\AXAutorun.log">
        <Run type="job" name="InitializeFMDataModel" />
</AxaptaAutoRun>
```

Now the application can be started from the Run... dialog box with the following command string: *ax32.exe -StartUpCmd=AutoRun_c:\fminitializedata.xml*.

You can incorporate this command line into a Launch AX And Set Start Location shared step along with some basic navigation so that the test case always starts from a known location, as shown in Figure 15-7.

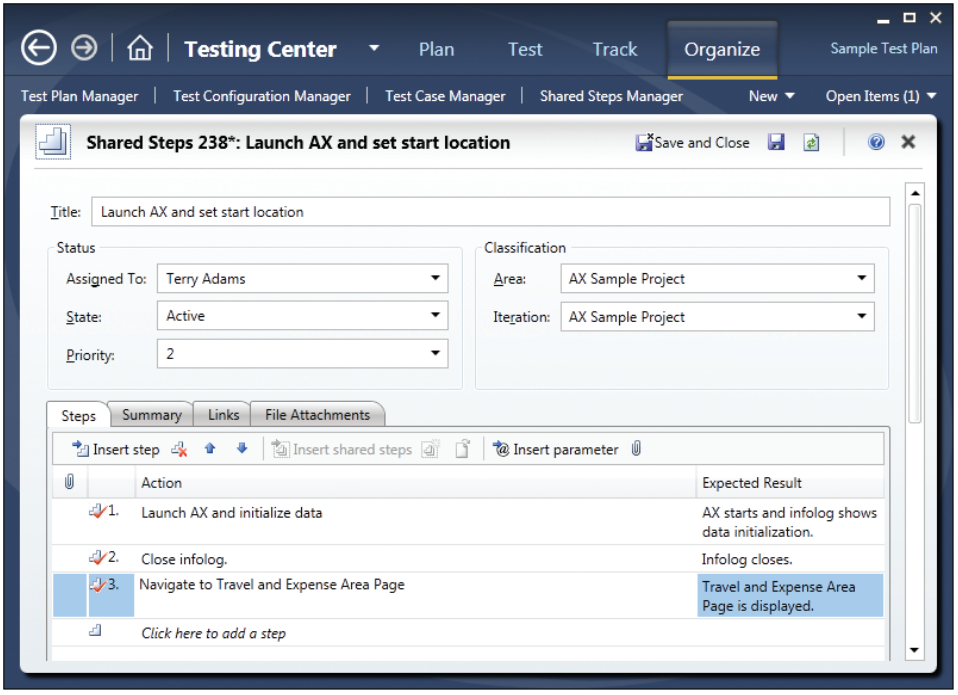


FIGURE 15-7 Shared step.

Record shared steps for fast forwarding

Microsoft Test Manager includes basic record-and-playback capability. Because of the focus on the manual tester, record and playback is not intended for end-to-end, push-button automation where a test is fully automated. Instead, a manual tester can use this functionality to "fast-forward" through the routine portion of a test case to get to the interesting part, where validation is required and exploratory testing drives the effective discovery of bugs.

Shared steps are a great starting point for building efficiency into your manual testing through fast-forward capabilities. Microsoft Test Manager can record a shared step independently (Organize > Shared Steps Manager > Create Action Recording). The actions recorded for the shared step can be used in all test cases that use those actions. You can also optimize the playback by using the most efficient and reliable actions.

With its long command line and the need to consistently be in a known state, the Launch AX And Set Start Location shared step shown in Figure 15-7 is a great candidate to record. To make this as efficient and reliable as possible, you can perform the following steps:

- To get to the Run dialog box to type in the command line, use **Windows logo key+R** instead of the mouse to open it. In general, shortcut keys are a better option for record and playback.
- To navigate to a particular area page, type the path into the address bar. This approach has multiple advantages because it goes directly to the page and is independent of the location where the application was last left.

The left side of Figure 15-8 shows what Microsoft Test Manager looks like after recording the shared step. The right side of Figure 15-8 shows a test case using the shared step. Notice the green arrow in the highlighted first step. This gives you the option to fast-forward through the shared step.

Develop test cases in an evolutionary manner

Creating detailed, step-by-step test cases early in the development process can become counterproductive as the application evolves to its final form. A better alternative is to develop your test cases in phases:

- **Phase 1** Identify the titles of the test cases needed for the requirements planned in your current development phase. Create the test cases and associated metadata (area, priority, and so on).
- **Phase 2** Flesh out the test case using an intent-driven approach. Perhaps a test case requires the creation of a customer with a past due account. Performing these actions require many steps. Starting with the Create Customer With Past Due Account step is sufficient in this phase.
- **Phase 3** Add details to the test cases as required. If your testers are domain experts, you may not need additional details. While omitting details introduces variability, the action recording provides the developer with the details of the steps taken.

  This phase also provides an opportunity to create additional shared steps that can be reused across test cases. If multiple test cases require Create Customer With Past Due Account, create a shared step and possibly record it. Alternatively, you can include an appropriate customer record in your data.

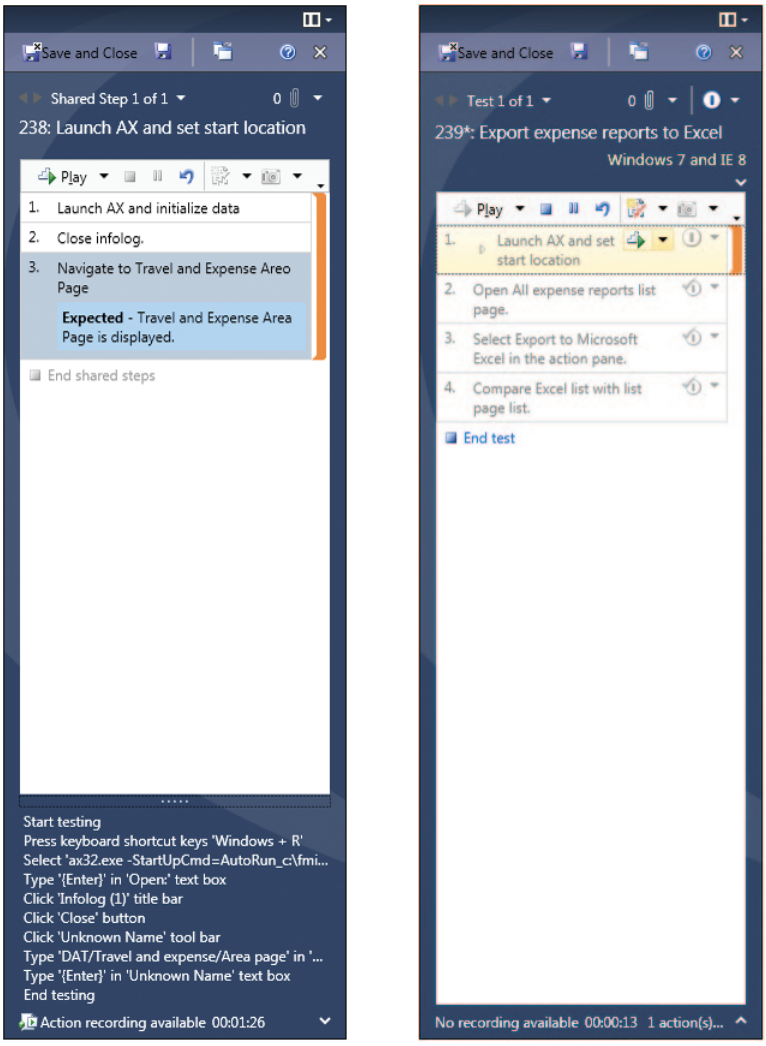


FIGURE 15-8 Microsoft Test Manager showing a recorded step and a test case that uses it.

Use ordered test suites for long scenarios

Scenario tests are valuable for business applications because long workflows are typical of business processes. Mapping these long scenarios to a test case can be challenging because you don't want a test case that has dozens of steps.

Microsoft Test Manager solves this problem by providing the capability to define the order of test cases within a test suite. Figure 15-9 shows an example of a human resources end-to-end scenario that is divided into many short test cases and then ordered within a test suite.

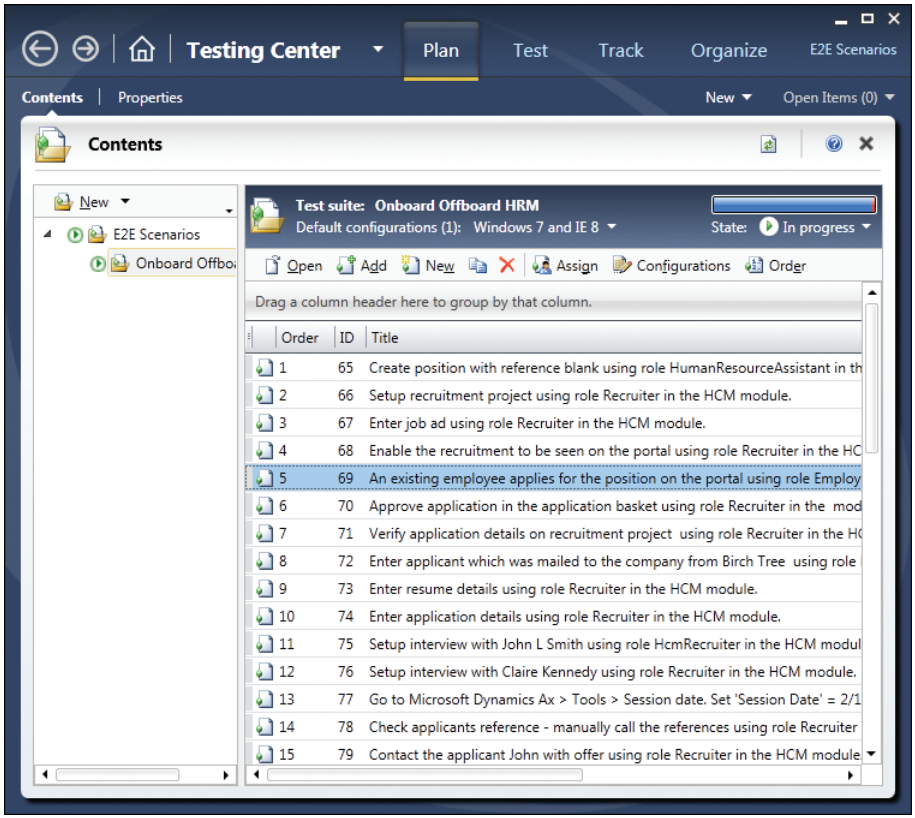


FIGURE 15-9 Test suite with multiple test cases.

Putting everything together

So far, this chapter has discussed some key aspects of developer testing and functional testing. This section ties these topics together with some bigger-picture application lifecycle management areas.

Execute tests as part of the build process

The Visual Studio 2010 ALM solution also includes Team Foundation Build, a workflow-enabled system that you use to compile code, run associated tests, perform code analysis, release continuous builds, and publish build reports. You can apply Team Foundation Build to Microsoft Dynamics AX projects. While the build process is beyond the scope of this chapter, running tests from Team Foundation Build is not.

Tests that are executed as part of a build process must be fully automated. Given this requirement, the starting point should be tests written using the SysTest framework. Fortunately, some tools are in place to enable execution of Microsoft Dynamics AX SysTest test cases from the Visual Studio environment.

The first step is for the SysTest framework to provide results in a format that Visual Studio expects, specifically the TRX output format. There are two pieces of good news here. First, the SysTest framework provides an extensible model for test listeners for results. Second, the Microsoft Dynamics AX partner ecosystem has provided a sample TRX implementation on CodePlex using the Test Listeners capability. The SysTestListenerTRX package for Microsoft Dynamics AX 2012 can be downloaded from *http://dynamicsaxbuild.codeplex.com/releases*.

The second step is to be able to initiate tests from Visual Studio. The Generic Test Case capability was developed to wrap an existing program. This is perfect for this situation because Microsoft Dynamics AX can run a test project from the command line and specify the Test Listener and the location of the output file.

Suppose you want to execute all tests marked with *SysTestIntegrationTestAttribute* that were created earlier in this chapter. After downloading and installing the SysTestListenerTRX package from the link shown earlier, do the following:

1. Create a new test project in Microsoft Dynamics AX. Add all test classes that have *SysTestIntegrationTestAttribute* on the class or on a method. As described for check-in tests earlier in this chapter, right-click the project, and then click Settings. In the Settings window, select Integration Tests.
2. Create a new test project in Visual Studio.
3. On the Test menu, click New Test, and then, in the Add New Test dialog box, double-click Generic Test.
4. Set up the generic test as shown in Figure 15-10, by completing the following steps:
   - In Specify An Existing Program... type the full path to *Ax32.exe*.
   - In Command Line Arguments... type the string shown in Figure 15-10. This string specifies that the Dynamic AX test project named "IntegrationTests" should be run, that the TRX listener is used, and that the output will be placed in *%TestOutputDirectory%\AxTestResults.trx*.
   - Under Results Settings, select the Summary Results File check box, and then specify the location for the results by using the same path and name as on the command line.

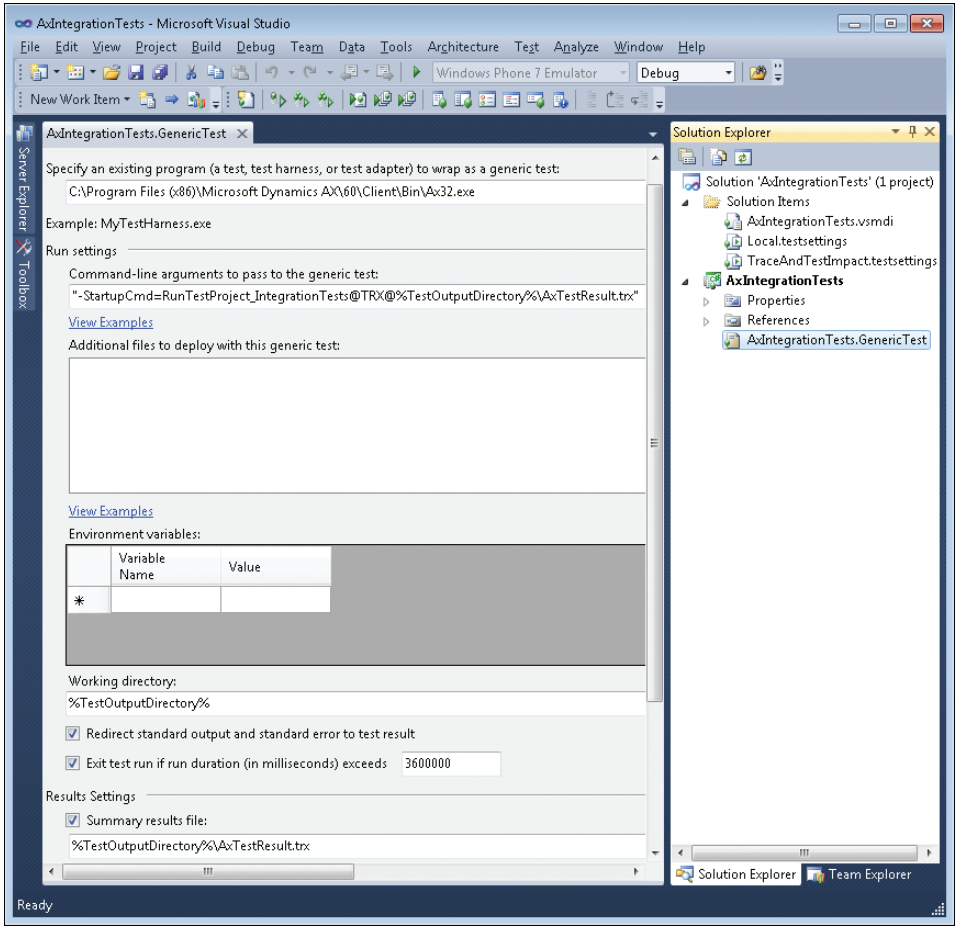


FIGURE 15-10 Test settings.

When you run all tests in the solution from Visual Studio (Click the Test menu > Run > All Tests In Solution), you will see the Microsoft Dynamics AX client open and then close. The results for this example are shown in Figure 15-11. Two tests were marked with *SysTestIntegrationTestAttribute,* and both passed.

Use the right tests for the job

A typical Microsoft Dynamics AX development project has four unique environments: development, test, preproduction, and production. This section provides a brief description of each environment and discusses how to apply the test tools in this chapter to each of them.

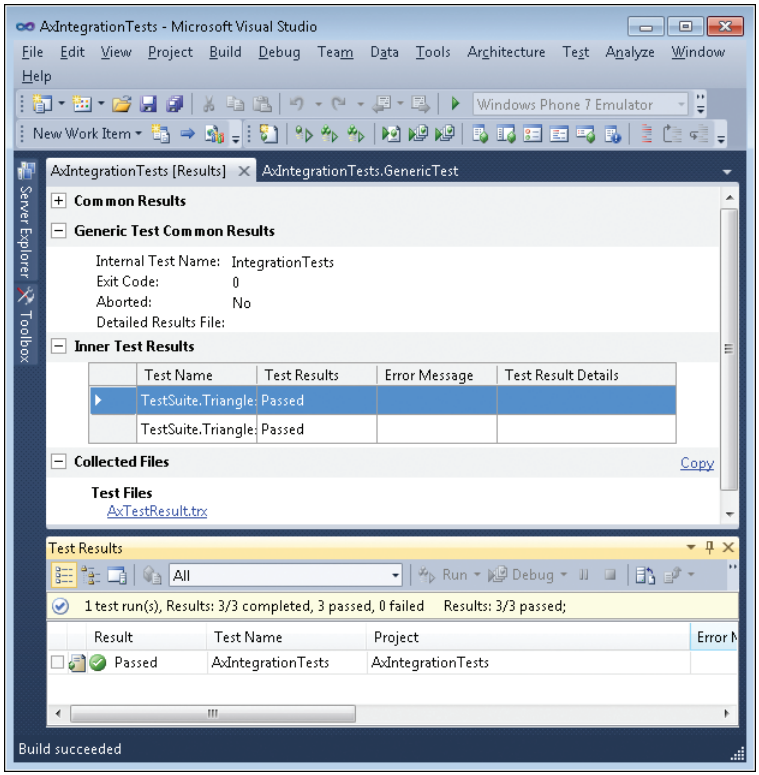


FIGURE 15-11 Test results.

The development environment is where developers are actively contributing code. A quality development process focuses on ensuring quality as close to the source of possible defects as possible. This is an excellent opportunity to use the SysTest framework for developing unit tests for new classes or methods and integration tests for basic multi-class interaction. Over time, these automated tests can form a regression suite that can be executed during the check-in and build processes as described in this chapter. The ATDD process described earlier in this chapter for validating requirements should also be applied in the development environment, so the testers on the project need to be involved during the development phase, optimally using the Visual Studio 2010 test tooling.

Broader testing is targeted for the test environment. Varying levels of integration testing are typical of this environment, with a strong focus on ensuring that business processes are functioning end to end. Creating test suites that use ordered test cases in Microsoft Test Manager is a good approach here. This is a good opportunity to evolve the detail of the test cases, using a well-designed approach for shared steps to minimize duplication across the suites. As the product changes and new builds are created, the SysTest regression suite should continue to be executed.

User acceptance testing (UAT) is the primary activity in the preproduction environment. The Microsoft Test Manager test suites developed for the test environment can form the basis for the UAT performed by users in the business. The data that you use for this testing should be a snapshot of the production data.

If all goes well in the previous environments, the code is deployed to production. To minimize downtime, only a cursory check, or smoke test, is performed after the new code is deployed. This typically is a manual test case defined by the business but exercised by the IT specialists performing the deployment. Once again, you can use Microsoft Test Manager to define the test case and provide an execution environment with recorded steps for auditing and debugging purposes.

To ensure that you have a quality, comprehensive test plan in place, you may want to review additional documentation that contains processes and guidelines for quality-focused development. For more information, see the Microsoft Dynamics AX 2012 white paper "Testing Best Practices," available at *http://www.microsoft.com/download/en/details.aspx?id=27565,* and Microsoft Dynamics SureStep methodology, at *http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=5320*.

CHAPTER 16

Customizing and adding help

In this chapter

Introduction 545
Help system overview 546
Help content overview 549
Create content 550
Publish content 567
Troubleshoot the Help system 572

Introduction

Microsoft Dynamics AX 2012 introduces a new Help system that was designed to make it easier for customers and partners to create and publish custom Help.

In previous versions, the Help system consisted of .chm files that were installed individually on each client computer. To customize Help, you had to decompile the .chm file, create custom .html files, recompile the .chm file, and then reinstall the .chm file on each client computer. Customizations were overwritten by Help updates from Microsoft.

The new Help system solves these problems. Help content is installed once on a server and displayed in a viewer on each client. Help topics consist of .html files, so you don't have to decompile and recompile .chm files. Although not required, separate folders that you create on the Help server can prevent customizations from being overwritten by Help updates from Microsoft.

You can customize the new Help system in the following ways:

- Create new topics in any HTML editor, either from scratch or by using the templates that are included. You can give your topics a consistent look and feel by applying the same style sheet that is used for the Help system in Microsoft Dynamics AX.
- Include references to user interface labels, fields, and menu items to ensure that your Help topics match the customizations that you've made to the user interface. Also, by using menu items, readers can open forms in Microsoft Dynamics AX directly from your Help topics.
- Make your Help context-sensitive, so that your topic appears when a user presses F1 on a specific object in the user interface.

- Replace an existing topic with a customized version of the topic, or display the customized topic alongside the existing one. You can display topics from multiple publishers or suppress topics from specific publishers.
- Create a table of contents for your topics and append it to the default table of contents. You can add context-sensitive Help that appears when you press F1. You can also apply search keywords to your topics to make them more discoverable.

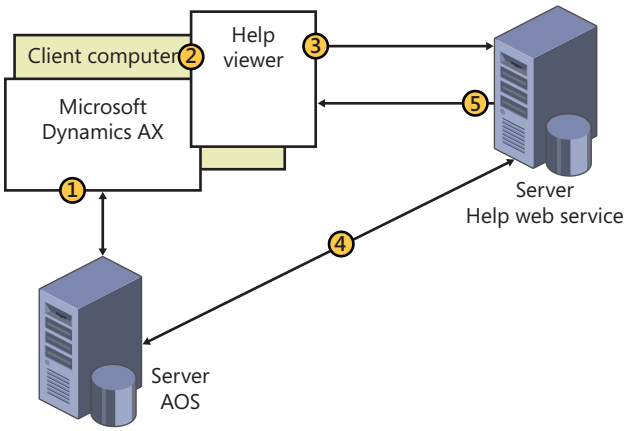
Note The Help system supplies Help only for the Microsoft Dynamics AX Windows client, not the Enterprise Portal web client.

Help system overview

The Help system consists of these components:

- **Help server** A centralized web service that responds to requests for Help documentation. You put your custom Help files on the Help server.
- **Help viewer** An application that is installed with the Microsoft Dynamics AX client. The Help viewer displays topics when a user requests help from the application.

These components interact with the Microsoft Dynamics AX client and Application Object Server (AOS) to display Help topics. Figure 16-1 describes the sequence of events between a request for a Help topic and the display of the topic.



1. The Microsoft Dynamics AX client retrieves the URL of the Help server.
2. The client calls the Help viewer.
3. The Help viewer requests the specified Help topic ID from the Help server.
4. The Help server retrieves values for the labels in the Help topic.
5. The Help server sends the topic to the Help viewer, which then displays the topic.

FIGURE 16-1 How the Help system works.

The following sections describe the components of the Help system in detail and explain how a request for a Help topic is processed.

Microsoft Dynamics AX client

When a user presses F1 or clicks the Help button in a form, the client performs the following actions:

1. The client identifies the Help topic to retrieve. The client obtains the ID of the form that is open when the user presses F1.
2. The client retrieves the URL of the Help server. The first time that a user requests help, the client contacts the AOS to retrieve the URL of the Help server. The client then caches the URL so that it can be used for additional Help requests.
3. The client calls the Help viewer. If the Help viewer is not already running, it starts. The call to the Help viewer includes the URL of the Help server and the ID of the form.

Help viewer

A user can click a link in the Help viewer to request a topic, or the user can search for topics. The Help viewer contacts the Help server and then retrieves and displays the specified topic.

If the Help server finds multiple topics for the specified ID, it displays a list of links to the topics on a summary page. If the user searches, the Help viewer lists links to the topics that are found. Figure 16-2 shows the Help viewer, which was designed to have the familiar look and feel of a web browser. The table of contents is displayed in the left pane and the Help topic is displayed in the right pane.

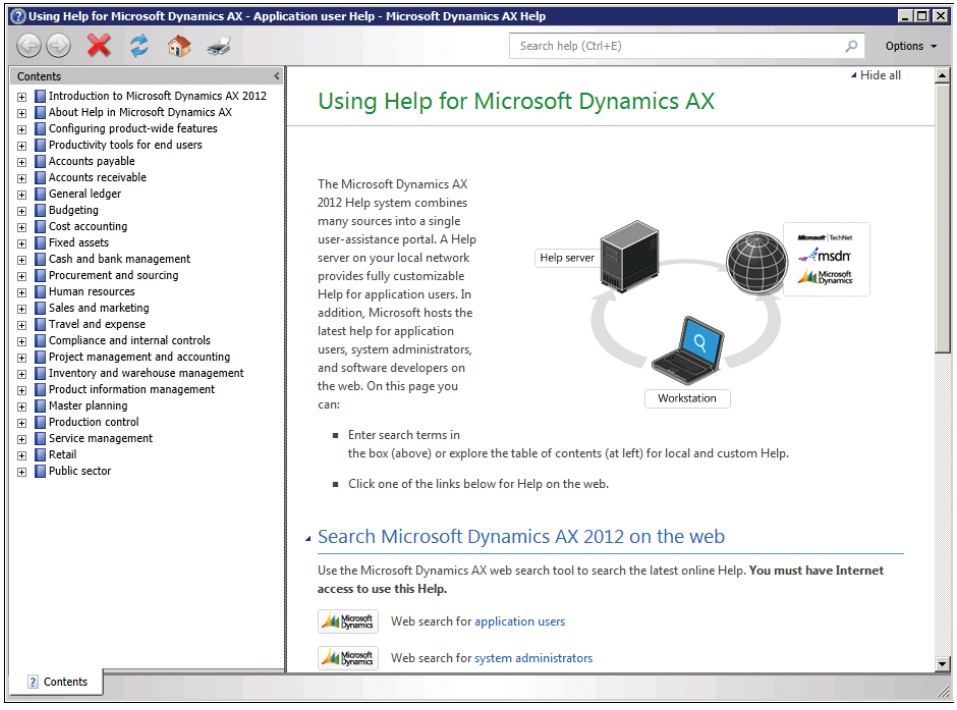


FIGURE 16-2 The Microsoft Dynamics AX Help viewer.

Help server

The Help server has the following components that respond to requests from a Help viewer.

Help web service

The Help web service is an Internet Information Services (IIS) web server application that responds to Help viewer requests for Help topics. The Help web service receives the request, finds the topic that matches the request, retrieves the text for the topic's labels from the AOS, and then sends the topic to the Help viewer.

Document files

Document files consist of XML and HTML files that are installed on the web server.

The XML files contain information for the table of contents that appears in the Help viewer.

Each HTML file contains a Help topic that appears in the Help viewer. Each HTML file also includes properties that uniquely identify the topic and provide additional information, such as the language and keywords, which aid in searches. These properties must be set properly for the topic to appear and be ranked appropriately in search results. When responding to a request, the Help web service searches for documents whose properties match the criteria sent by the Help viewer.

Document files are installed in a folder structure on the Help server. The default location is C:\inetpub\wwwroot\DynamicsAX6HelpServer\content, but this location can be changed during installation.

Each organization or individual that creates and publishes content for the Help system is called a *publisher*. Upon installation, the Help system contains a folder for a single publisher: Microsoft. When you add topics to the Help system, you create a new folder structure beneath the content folder to hold your document files—for example:

- C:\inetpub\wwwroot\DynamicsAX6HelpServer\content\Microsoft
- C:\inetpub\wwwroot\DynamicsAX6HelpServer\content\YourFolder

Each document file belongs to a *document set*, which is a named collection of related Help topics. You use document sets to associate a collection of Help documents with either the client or the Development Workspace. The Help system includes the following document sets:

- **ApplicationHelpOnTheWeb** Provides Help on the web for users of the Microsoft Dynamics AX client. You cannot add new documents to this document set.
- **DeveloperDocumentation** Provides Help on the web for users of the Development Workspace. You cannot add new documents to this document set.
- **Glossary** Provides glossary entries for users of Microsoft Dynamics AX. You can add new documents to this document set.

- **SystemAdministratorHelpOnTheWeb** Provides Help on the web for system administrators of Microsoft Dynamics AX. You cannot add new documents to this document set.
- **UserDocumentation** Provides Help for users of the Microsoft Dynamics AX client. When you create custom topics, you add them to this document set.

Windows Search Service (WSS)

Installing the Help web service enables Windows Search Service (WSS), which indexes the document files that are added to the Help server. The index includes the document properties of each HTML file.

When the Help web service receives a request, it queries the WSS to find the document files that match the criteria specified by the request. The Help web service uses the following order of precedence to match and rank the search results that are displayed in the Help viewer:

1. **Keywords** Matches the search request to keywords for the topic
2. **Title** Matches the search request to part of the title of the topic
3. **Topic ID** Matches the search request to the ID that uniquely identifies the topic
4. **Content** Matches the search request to one or more values found in the content of the topic

AOS

To support the Help system, the AOS performs the following actions:

- Stores the URL of the Help server. Each Help viewer retrieves this URL before sending a request for content, so that changes to the URL are available to all clients of the AOS.
- Returns the text associated with a label. The Help web service retrieves label text and adds that text to the HTML of the topic. This ensures that the text in the content matches the text in the user interface.

Help content overview

This section describes the concepts and components that are involved in customizing the Help system.

Topics

A *topic* is the content for a specific subject area. Microsoft Dynamics AX Help is organized by topic. Topic files are HTML files, and each topic has a unique ID. When you plan your customization, evaluate how your changes fit into the existing topic structure. You can either add topics or update topics.

Add a topic when you want to document a new process, form, or other component. You should add entries for new topics to the table of contents. For a context-sensitive topic, the topic ID must match the ID of the form or other component that you are documenting.

Update a topic when you want to document a change to an existing process, form, or other component. An updated topic supplements or replaces an existing topic. When you update a topic, your content must include the same topic ID as the existing content. For more information, see "Update content from other publishers," later in this chapter.

Caution Do not edit or delete any files that were created by Microsoft or any other publisher. If you change an existing file, your changes might be lost during an update or reinstallation of the documentation from that publisher.

Publisher

A *publisher* is an individual or organization that has documentation on the Help server. Each content element includes a document property that specifies the ID of a publisher. The publisher ID is one of the document properties that you can use to replace documentation for an existing topic. The content from each publisher is organized in its own folder on the Help server.

Table of contents

The *table of contents* file is an XML file that contains a hierarchical list of topics that is displayed in the left pane of the Help viewer (see Figure 16-2). Each entry in the table of contents is a link to a topic.

You can add entries to the table of contents when you want your topic to be more easily discovered and viewed from the Help viewer. If you have several related topics, you can use the table of contents to display the topics in a hierarchical group.

Summary page

A *summary page* is a list that the Help viewer displays when the requested content includes more than one topic. Figure 16-3 shows an example of a summary page. To view a specific topic, the user clicks the link for that topic.

Create content

Before you write a new topic or update an existing topic, use these guidelines to plan your work:

- Decide what topics your documentation requires and what documents you have to include.
- The Help server requires all topics to use the Extensible Hypertext Markup Language (XHTML) standard.

- If you are updating an existing topic, determine the ID of the topic. If you are adding a new topic, decide whether to add an entry to the table of contents. Later sections in this chapter describe how to update existing contents and add topics to the table of contents.
- Gather the information for the document properties that are required to identify the content. For more information, see the following section, "Create a topic in HTML."

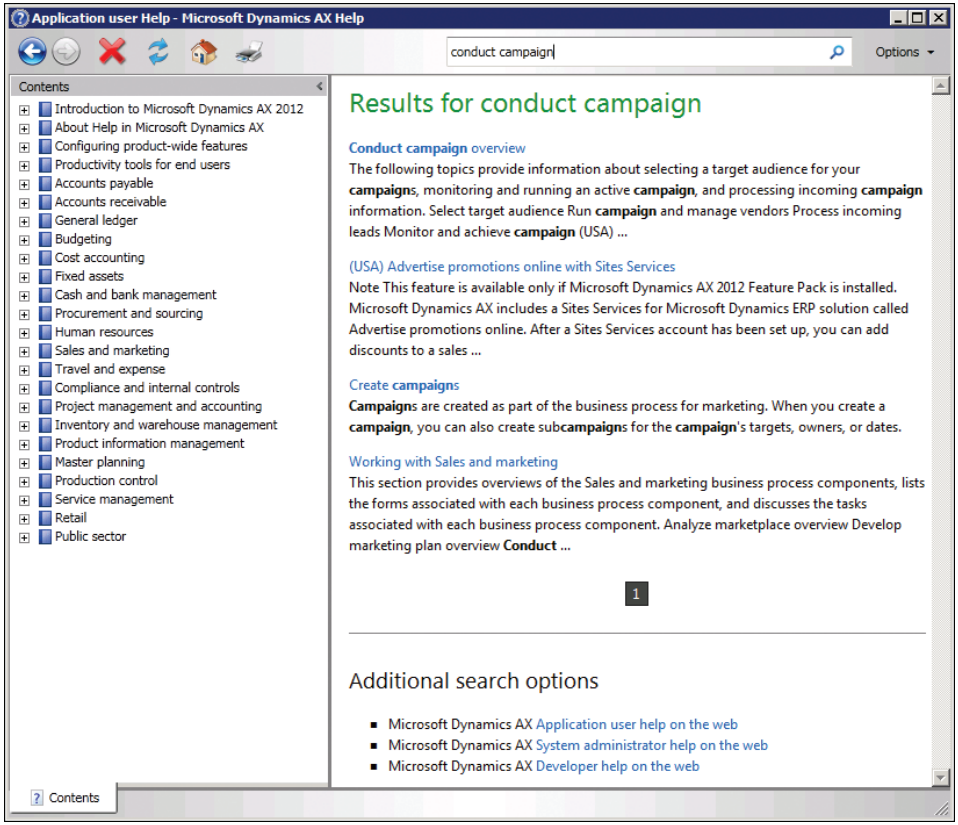


FIGURE 16-3 The Microsoft Dynamics AX Help summary page.

To quickly create documentation that matches the look of Microsoft Dynamics AX Help, you can use the templates that are included with the Help system. Each template contains a framework of elements, styles, and guidelines that can make creating content faster and simpler. To see the list of the templates, open the Help viewer, type **Templates for Help Documentation** in the search box, and then press Enter. There are templates for the following types of files:

- HTML templates that resemble the Help documentation from Microsoft. These templates represent common topic types, such as orientation topics, procedure topics, key task topics, and form topics. The HTML templates are an excellent option if you are creating new topics. (If you are reusing HTML topics that already exist, you can publish them on the Help server so long as you add the correct metadata. For more information, see the following section, "Create a topic in HTML.")

- A Microsoft Word template that can be used to create documentation with Word 2007 or a later version. Typically, a super user within an organization, such as an office manager, will use the Word template to publish organization-specific guidelines and processes related to work that users perform in Microsoft Dynamics AX. The Word template includes the capability to create the supplemental HTML file that is required for each Word file. For more information see the section "Create non-HTML content," later in this chapter

Create a topic in HTML

This section describes how to create an HTML file from scratch. You can use any HTML or text editor to create HTML files. For example, if you are using Microsoft Visual Studio, you would create a text file. In order for the file to appear in the Help viewer and look consistent with the Help that Microsoft provides, you'll need to add specific metadata.

The following sections walk you through the process of creating a topic that contains the correct references and metadata.

Declarations

After you first create the HTML file, use the information in this section to add the initial elements and metadata for the topic. When you complete this section, you will have a basic HTML document.

1. Add a <*doctype*> element, and specify the document type definition. The following table shows the document type declarations to use:

Declaration	Value
TopElement	*html*
Availability	*PUBLIC*
Document type definition	*-//W3C//DTD XHTML 1.0 Transitional//EN*
URL	*http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd*

   The following HTML shows the declarations in the <*doctype*> element:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"[]>
```

2. Add an <*html*> element, add the *dir* attribute, and then set the attribute value to *"ltr"* (left-to-right).

   Although the *dir* attribute is not required, the Help viewer uses its value to optimize the appearance of the document:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"[]>
<html dir="ltr">
</html>
```

3. Add the namespaces in the following table to the <*html*> element.

Important These namespaces are required. If you do not include every namespace, your document might not appear in the Help viewer.

Namespace	URL/URN
xmlns:xlink	*http://www.w3.org/1999/xlink*
xmlns:dynHelp	*http://schemas.microsoft.com/dynamicsHelp/2008/11*
xmlns:dynHelpAx	*http://schemas.microsoft.com/dynamicsHelpAx/2008/11*
xmlns:MSHelp	*http://msdn.microsoft.com/mshelp*
xmlns:mshelp	*http://msdn.microsoft.com/mshelp*
xmlns:ddue	*http://ddue.schemas.microsoft.com/authoring/2003/5*
xmlns:msxsl	*urn:schemas-microsoft-com:xslt*

The following HTML shows the namespace declarations:

```
<html DIR="LTR" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:dynHelp="http://schemas.microsoft.com/dynamicsHelp/2008/11"
xmlns:dynHelpAx="http://schemas.microsoft.com/dynamicsHelpAx/2008/11"
xmlns:MSHelp="http://msdn.microsoft.com/mshelp"
xmlns:mshelp="http://msdn.microsoft.com/mshelp"
xmlns:ddue="http://ddue.schemas.microsoft.com/authoring/2003/5"
xmlns:msxsl="urn:schemas-microsoft-com:xslt">
```

4. Add the HTML head and body elements to the document. The following example shows the HTML:

```
< <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"[]><html DIR="LTR"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:dynHelp="http://schemas.microsoft.com/dynamicsHelp/2008/11"
xmlns:dynHelpAx="http://schemas.microsoft.com/dynamicsHelpAx/2008/11"
xmlns:MSHelp="http://msdn.microsoft.com/mshelp"
xmlns:mshelp="http://msdn.microsoft.com/mshelp"
xmlns:ddue="http://ddue.schemas.microsoft.com/authoring/2003/5"
xmlns:msxsl="urn:schemas-microsoft-com:xslt">
   <head>
   </head>
   <body>
   </body>
</html>
```

Document head

The document head contains metadata, plus the document title that appears in the title bar of the Help viewer. The style sheets that you reference in the document head are the same style sheets referenced by the Help provided by Microsoft. By using these style sheets, you can give your documentation a look and feel that is consistent with the Microsoft Help.

1. Add two *<meta>* elements, and then set their attributes as follows:

   - In the first *<meta>* element, set the *http-equiv* attribute to *"Content-Type"*, and then set the *content* attribute to *"text/html; charset=UTF-8"*.

- In the second <*meta*> element, set the *name* attribute to *"save"*, and then set the *content* attribute to *"history"*.

The following example adds the <*meta*> elements. Notice how the first <*meta*> element specifies the document type. Also notice how the second <*meta*> element specifies that the document is saved to the session memory of the browser:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<meta name="save" content="history"/>
```

2. Add a <*title*> element. This text appears in the title bar of the Help viewer as shown in Figure 16-4.

```
<title>Using Help for Microsoft Dynamics AX</title>
```

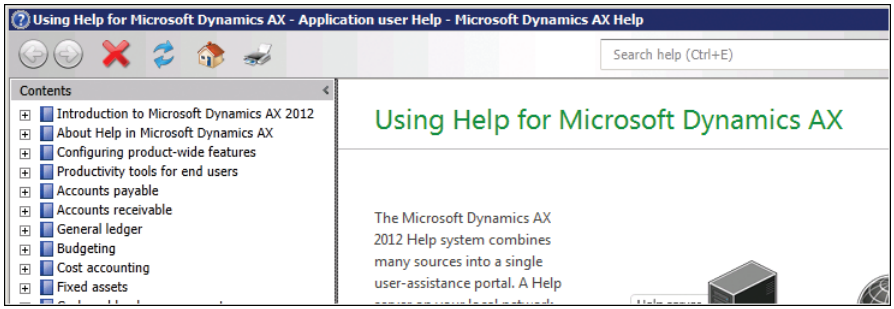


FIGURE 16-4 Microsoft Dynamics AX Help topic title.

3. Add three <*link*> elements, and then use them to specify the style sheets to apply to your document:

- Add a *rel* attribute to each <*link*> element, and then set the value of each to *"stylesheet"*.
- Add a *type* attribute to each <*link*> element, and then set the value of each to *"text/css"*.
- Add an *href* attribute to each <*link*> element, and then use the following table to specify the value of each:

Style Sheet	Description
AX.css	Specify a path from the folder on the Help server where you publish the document to the folder that contains the *AX.css* file. By default, the file is installed in the following folder: *C:/inetpub/wwwroot/<HelpServerName>/content/Microsoft/EN-US/local*. To use the relative path for the default installation folder, type **../local/AX.css.**
presentation.css	Specify a path from the folder on the Help server where you publish the document to the folder that contains the *presentation.css* file. By default, the file is installed in the following folder: *C:/inetpub/wwwroot/<HelpServerName>/content/Microsoft/EN-US/local*. To use the relative path for the default installation folder, type **../local/presentation.css.**
HxLink.css	Specify the URL of the of the *HxLink.css* file. To use the default location, type **ms-help://Hx/HxRuntime/HxLink.css.**

In the following example, notice how the *href* attributes specify the relative paths of the folders that contain the .css files, assuming that the content is published to the appropriate publisher and language folders. Also, the third <*link*>element specifies a URL for the *HxLink.css* file:

```
<link rel="stylesheet" type="text/css" href="../../Microsoft/EN-US /local/
presentation.css"/>
<link rel="stylesheet" type="text/css" href="../../Microsoft/EN-US /local/AX.css"/>
<link rel="stylesheet" type="text/css" href="ms-help://Hx/HxRuntime/HxLink.css"/>
```

4. Add nine <*meta*> elements, and then provide the required document properties. Add a *name* and *content* attribute to each element. For the *name* and *content* attribute of each element, specify values in the following table:

Name	Content
Title	The title of the topic; for example, "Using Help for Microsoft Dynamics AX."
Microsoft.Help.Id	A unique ID for the topic. The ID value must be unique and cannot duplicate the ID of an existing topic. Typically, you use a globally unique identifier (GUID) to ensure that the ID value is unique, but you can use a text value for the ID.
ms.locale	The locale for the Help language; for example, EN-US.
publisher	The name of the publisher, generally the name of your company; for example, Contoso. For more information, see the section "Publish content," later in this chapter.
documentSets	The name of the default document set: *UserDocumentation*. A Help topic can belong to multiple document sets.
Microsoft.Help.Keywords	Provide a semicolon-delimited list of keywords that will help users find your topic. For example, Contoso; customer.
suppressedPublishers	Leave the *content* attribute empty.
Microsoft.Help.F1	If you want the Help to be context-sensitive, add the ID of the element that you want to associate with the topic. For example, in Microsoft Dynamics AX, CustTable is the name of the Customers form. To create the *Microsoft.Help.F1* element, add "Forms" and a period to the form name (Forms.CustTable). If you want to call the same Help topic from more than one form, add the ID for each element, separated by a semicolon. For more information, see the section "Make a topic context-sensitive," later in this chapter.
Description	Provide a brief description of the topic.

In the following example, notice how the *name* and *content* attributes specify each document property and its value:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta name="save" content="history" />

<title>Contoso customer help information</title>

<link rel="stylesheet" type="text/css" href="../../Microsoft/EN-US /local/
presentation.css"/>
<link rel="stylesheet" type="text/css" href="../../Microsoft/EN-US /local/AX.css"/>
<link rel="stylesheet" type="text/css" href="ms-help://Hx/HxRuntime/HxLink.css"/>

<meta name="Title" content="Contoso customer help information"/>
<meta name="Microsoft.Help.Id" content="Contoso.Forms.CustTable"/>
<meta name="ms.locale" content="EN-US"/>
```

```
<meta name="publisher" content="Contoso"/>
<meta name="documentSets" content="UserDocumentation"/>
<meta name="Microsoft.Help.Keywords" content="Contoso; customer"/>
<meta name="suppressedPublishers" content=""/>
<meta name="Microsoft.Help.F1" content="Forms.CustTable"/>
<meta name="description" content="Describes Contoso customization to the Customers
form"/>
```

Document body

Between the open and close tags of the <*body*> element that you added earlier, add elements for controls, the document title, the main section, and links to related topics.

1. Add <*input*> elements for two hidden controls that the Help viewer uses to display topics. These controls are required.

   Add a *type*, *id*, and *class* attribute to each element. Use the following values for the attributes:

Type	ID	Class
hidden	*hiddenScrollOffset*	Not applicable (This control does not require a class attribute.)
hidden	*userDataCache*	*userDataStyle*

   In the following example, notice how the *type* attributes specify that each control is hidden, and the *id* attributes specify each control name. Also notice how the *class* attribute of the "*userDataCache*" control is set to "*userDataStyle*":

```
<input type="hidden" id="userDataCache" class="userDataStyle" />
<input type="hidden" id="hiddenScrollOffset" />
```

2. Add the document header. Add a <*div*> element, and then set the *id* attribute to "header", as shown in Step 4.

3. Add two <*span*> elements to the header section:

   - For the first one, set the *id* attribute to *"runningHeaderText"*.
   - For the second, set the *id* attribute to *"nsrTitle"*. To specify the title that appears in the Help viewer, type a title for the topic ("Contoso customer Help information" in the following example).

4. Add an <*hr*> element. To keep the title visible during scrolling, set the *class* attribute to *"title-divider"*. Notice that the <*br*> element adds an empty line above the title:

```
<div id="header">
   <br />
   <span id="runningHeaderText"> </span>
   <span id="nsrTitle">Contoso customer Help information</span>
   <hr class="title-divider" />
</div>
```

5. Add the main section. Add a <*div*> element, and then set the *id* attribute to *"mainSection"*.

6. Add a <*div*> element to the main section, and then set the *id* attribute to *"mainBody"*.

7. Add another <*div*> element, and then set the *id* attribute to *"footer"*:

```
<div id="mainSection">
   <div id="mainBody">
   </div>
   <div id="footer">
   </div>
</div>
```

Content

In this section, add all the elements between the start and end tags of the main body. The following sections add an introduction, a description of a customization, and a list of links to related topics.

1. Add an introduction. Add a <*div*> element, set the *class* attribute to *"introduction"*, and then type an introduction for your topic within one or two <*p*> (paragraph) elements.

Important The introduction is a required element. When a user searches for a topic, the introduction is used as an abstract on the summary page that displays search results. (See Figure 16-3.)

```
<div class="introduction">
   <p>
      This topic includes information about changes to the Customer form that have been
added by Contoso.
   </p>
   <p>
      You can use the Customer form to view additional information about each of your
customers.
   </p>
</div>
```

2. Create a section heading by adding an <*h1*> element, and then set the *class* attribute to *"heading"*. Between the start and end tags, type a heading for the section. If you do not want to use a heading for the section, you can leave the <*h1*> element empty, as in the example.

3. Add a <*div*> element, and then set the *class* attribute to *"section"*.

4. Add your Help content. You can add standard HTML tags to format your content. For example, you can use paragraphs, sections, headings, bulleted lists, ordered lists, tables, and formatting such as bold and italic. The following example shows the opening paragraph for a section:

```
<h1 class="heading"></h1>
<div class="section">
   <p>
```

```
      The Contoso customer add-ons enable you to view important information about your
   relationship with your customer. To view the additional information, click one of the
   following buttons:
      </p>
   </div>
```

Links to related topics

Most topics provided by Microsoft contain a "See also" section, which contains links to other topics that might help the user. Adding a "See also" section can help your custom documentation blend with the existing documentation from Microsoft. For information about how to find the ID of an existing topic to link to, see the section "Update content from other publishers," later in this chapter.

1. Create a section heading by adding an <*h1*> element, and then set the *class* attribute to *"heading"*. Type **See also** between the start and end tags.
2. Add a <*div*> element, and then set the *class* attribute to *"section"*.
3. Add a <*div*> element between the start and end tags of the <*div*> element, and then set the *class* attribute to *"seeAlsoStyle"*.
4. Add a <*span*> element between the start and end tags of the "See also" section.
5. Add a <*dynHelp*:topicLink> element between the start and end tags of the <*span*> element. Type the topic title that you want to link to as the text of the link. Add the following attribute values:

Attribute	Value
topicId	The ID of the topic that you want to link to.
documentSet	UserDocumentation

The following example creates a "See also" section with a link to a topic named "Create a customer account":

```
<h1 class="heading">See also</h1>
<div class="section">
   <div class="seeAlsoStyle">
      <span>
         <dynHelp:topicLink topicId="cc18943e-c00c-49e6-8bd2-03be6481b6dd" documentSet=
"UserDocumentation">Create a customer account</dynHelp:topicLink>
      </span>
   </div>
</div>
```

Footer

To give your topics a look that's consistent with existing Help topics, add a line to the footer section at the end of the document.

1. Add a <*div*> element between the start and end tags of the <*div*> element that has the *id* attribute set to "footer", and then set the *class* attribute to *"footerline"*.

2. Add an *<hr>* element, add the *style* attribute, and then set the following values:

Property Name	Value
height	*3px*
color	*Silver*

```
<div id="footer">
   <div class="footerLine">
      <hr style="height:3px; color:Silver" />
   </div>
   <p />
</div>
```

Add labels, fields, and menu items to a topic

You can enhance your Help by adding references to user interface labels. When you add a reference to a label, the label text is retrieved from the AOS and added to your Help topic at run time. This means that the user interface text that you refer to in your Help documentation will always match the text in the application.

The following table describes the types of labels that you can reference in your documentation:

Label Type	Description
Labels	Text that appears in the user interface. These types of labels are found in a Microsoft Dynamics AX label file.
Table fields	The user interface text that represents a field from a data table.
Menu items	The user interface text for a menu item.

The following restrictions can affect how labels appear in your documentation:

- To retrieve the text of the label, the Help server queries AOS. Whoever requests the Help topic must have access permissions. Otherwise, the default text appears in the topic.
- When the label appears in the Help viewer, the text appears in the same language that is used by the Microsoft Dynamics AX client that the request originated from.
- The Help server does not support references to labels in non-HTML documents.

Add a label from the user interface

When you create a Help topic that describes a form, you can include a specific label that appears in the form by adding the ID of the label to the HTML of your topic.

1. In the Development Workspace, point to Tools > Development Tools > Label > Label Editor.
2. In the Find What field, type the text of the label whose ID you want to find, expand the In The Language list, click the language you want, and then click Find Now.

   The Label Editor lists all the labels that include the specified text for the specified language.

3. In your Help topic, in the location where you want the label to appear, add a <*dynHelpAx*:label> element, and then specify values for the following attributes:

Attribute	Value	Description
axtype	*"Label"*	Set the attribute to **"***Label***"**.
id	The ID value of the label.	Specify the ID value that you retrieved by using the Label Editor.

```
<dynHelpAx:label axtype="Label" id="@SYS21829"> </dynHelpAx:label>
```

4. In the <*dynHelpAx*:label> element, specify default text. If the label cannot be retrieved, the text that you supply appears in the Help topic. In the following example, "Bank Account" is the default text:

```
<dynHelpAx:label axtype="Label" id="@SYS21829">Bank Account</ dynHelpAx:label>
```

Note If you do not supply a default text value, and the label cannot be retrieved, the Help topic will not include a value for the label.

Add a table field label

You can add table field labels when you want your Help topic to include the text of a table field.

1. In the Application Object Tree (AOT), under Data Dictionary\Tables, click the table that contains the field, and then note the value of the *Name* property of the table.
2. Expand *Fields*, find the field that contains the label that you want to use, and then note the value of the *Name* property of the field.
3. In your Help topic, in the location where you want the label to appear, add a <*dynHelpAx:label*> element, and then specify values for the following attributes.

Attribute	Value	Description
axtype	*"Field"*	Set the attribute to *"Field"*.
axtable	The name of the table.	Specify the value of the *Name* property of the table that contains the field.
axfield	The name of the field.	Specify the value of the *Name* property for the field.

```
<dynHelpAx:label axtype="Field" axtable="DirPartyTable" axfield="Name">
</dynHelpAx:label>
```

4. Specify a default text value for the <*dynHelpAx:label* > element. If the label cannot be retrieved, the text that you specify appears in the Help topic. In the following example, "Name" is the default text:

```
<dynHelpAx:label axtype="Field" axtable="DirPartyTable" axfield="Name">Name
</dynHelpAx:label>
```

Add a menu item label

You can add a menu item label to your Help topic when you want to include the text of a menu item.

1. In the AOT, expand Menu Items, and then expand a menu item category.
2. Find the menu item, and then note the value of the *Name* property.
3. In your Help topic, in the location where you want the label to appear, add a <*dynHelpAx*:*label*> element, and then specify values for the following attributes:

Attribute	Value	Description
axtype	"MenuItem"	Set the attribute to "*MenuItem*".
axmenutype	"Display"	Set the attribute to "*Display*".
axmenuitem	The name of the menu item.	Specify the value of the *Name* property for the menu item.

```
<dynHelpAx:label axtype="MenuItem" axmenutype="Display" axmenuitem="SalesTable">
</dynHelpAx:label>
```

4. Specify a default text value for the <*dynHelpAx*:label> element. If the label cannot be retrieved, the text that you supply appears in the Help topic. In the following example, "Sales order" is the default text:

```
<dynHelpAx:label axtype="MenuItem" axmenutype="Display" axmenuitem ="SalesTable">
Sales order</dynHelpAx:label>
```

Make a topic context-sensitive

Context-sensitive Help provides documentation for specific objects in Microsoft Dynamics AX. When a user presses F1, the Help viewer displays documentation about the form, list page, or other object that you have open in the Microsoft Dynamics AX client or the Development Workspace.

Microsoft Dynamics AX Help supports the use of context-sensitive Help for the following client and Development Workspace object types:

- Base enums
- Configuration keys
- Forms
- Maps
- Parts
- Tables
- Classes
- Data types

- List pages
- Menu items
- Reports
- Views

When a user presses F1, the following actions occur:

- The client sends the ID of each object that is currently open to the Help viewer. (The object ID is a string that identifies each object, such as *CustTable*.)
- The Help viewer sends the object IDs to the Help server.
- For each object ID, the Help server searches for content with a corresponding topic ID.
- When an object ID matches a topic ID, the Help server returns the content for that topic.
- The Help viewer receives and displays the content.

To make a topic content context-sensitive, you set the *Microsoft.Help.F1* property of your content to the ID of the object. If multiple topics have the same value for the *Microsoft.Help.F1* property, the Help viewer displays a list of multiple topic links that the user can choose among.

To find object IDs:

- In the AOT, right-click the object in the AOT, and then point to Add-Ins > Help Properties. The Help Properties window opens and displays the ID.
- In a form, open the form, right-click in a blank area of the form, click Personalize, and then click the Information tab. The Form name field displays the name of the form. To specify the ID, combine the element type, "Forms," with the name of the form. For example, for the form called CustTable, you would specify *Forms.CustTable* as the object ID, as in the following example:

```
<meta name="Microsoft.Help.F1" content="Forms.CustTable"/>
```

Update content from other publishers

At times, you might want to update or modify a topic that already exists on the Help server. For example, if your solution adds fields to an existing form, you might want to replace the default topic for that form with one of your own. To update an existing Help topic, you create a new topic to replace the existing one. The Help viewer then hides the existing topic by suppressing the publisher that you specify. The hidden topic remains on the Help server and can be accessed through search.

Caution Do not edit or remove any files that were published to the Help server by another publisher. An update or reinstallation of the files from that publisher might overwrite the changes that you make.

To replace a topic, you obtain metadata from the topic that you want to replace and then add it to the new topic that you've created.

1. In the Help viewer, open the topic that you want to replace, right-click the topic, and then click View Source.
2. Get metadata about the Help topic:
   - Search for *meta name="Microsoft.Help.F1"*, and then record the topic ID.
   - Search for *meta name="publisher"*, and then record the publisher ID.
3. In the new topic that you created, do the following:
   - Search for the element *<meta name="Microsoft.Help.F1" content=""/>*, and then set the value of *content* to the topic ID that you noted in step 2.
   - Search for the element *<meta name="suppressedPublishers" content=""/>*, and then set the value of *content* to the publisher ID that you noted in step 2.

Tip If you want to hide content from more than one publisher, use a semicolon to separate each publisher ID.

The following example sets the ID to the Microsoft topic to replace and adds "Microsoft" to the *<suppressedPublishers>* element:

```
<meta name="Microsoft.Help.F1" content="c3fc5774-6ed0-4760-86f5-7899e825ab25"/>

<meta name="suppressedPublishers" content="Microsoft"/>
```

4. Save the file, and then publish your content to the Help server.

Create a table of contents file

The table of contents file is an XML file that contains a hierarchical representation of Help topics. Add entries to the table of contents when you add new Help topics that you want to appear in the table of contents. (By convention, Microsoft Help topics that contain conceptual information and procedures appear in the table of contents, but topics that describe forms, which appear when the user presses F1, do not.) The table of contents file must be named *TableOfContents.xml*. After you create the XML file, you publish it. For more information about publishing your table of contents file, see the section "Publish content," later in this chapter.

1. Use a text or XML editor to create a new file.
2. Add the *<xml>* and *<tableOfContents>* elements to the file. The *<tableOfContents>* element requires XML namespace information.

```
<?xml version="1.0" encoding="utf-8"?>
<tableOfContents xmlns="http://schemas.microsoft.com/dynamicsHelp/2008/11" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
</tableOfContents>
```

3. Add metadata properties that specify a document set, language, and publisher.

The Help service uses these properties to identify the entries that appear in the Help viewer's table of contents. The following table describes the properties:

Property	Required	Description
documentSet	Yes	Specify the ID of a document set. Typically, you set this property to "*UserDocumentation*". The document set determines whether the entries appear in the table of contents for the application workspace, developer workspace, or both.
Ms.locale	Yes	Specify the language of the table of contents entries. Use a language code to identify the language. For example, use *EN-US* for U.S. English. The *ms.locale* property enables the Help viewer to display localized content.
publisher	No	Specify the ID of the publisher. Table of contents entries are grouped by publisher. For more information, see the section "Publish content," later in this chapter.

```
<?xml version="1.0" encoding="utf-8"?>
<tableOfContents xmlns="http://schemas.microsoft.com/dynamicsHelp/2008/11" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
   <publisher>Contoso</publisher>
   <documentSet>UserDocumentation</documentSet>
   <ms.locale>EN-US</ms.locale>
</tableOfContents>
```

4. Add the <*entries*> element after the metadata, as shown in the following example:

```
<publisher>Contoso</publisher>
<documentSet>UserDocumentation</documentSet>
<ms.locale>EN-US</ms.locale>
<entries>
</entries>
```

5. Add an <*entry*> element for each topic that you want to add to the table of contents:

The <*entry*> elements must be child elements of the <*entries*> element. The following table describes the properties of an <*entry*> element:

Property	Required	Description
text	Yes	Specify the text that appears in the Help viewer.
Microsoft.Help.F1	No	Specify one or more topic IDs that are associated with the entry. When you click an entry in the table of contents, the Help viewer uses these IDs to request the content elements associated with that entry.
publisher	No	Specifying the publisher is optional.

```
<entries>
   <entry>
      <text>Sample help topic</text>
      <Microsoft.Help.F1>DEFAULT_TOPIC</Microsoft.Help.F1>
   </entry>
</entries>
```

6. If you want entries to appear under the current entry in the Help viewer in a hierarchical structure, add a <*children*> element, and then add entries to it:

```
<entry>
   <text>Sample help topic</text>
   <Microsoft.Help.F1>DEFAULT_TOPIC</Microsoft.Help.F1>
   <children>
      <entry>
         <text>Child help topic</text>
         <Microsoft.Help.F1>DEFAULT_TOPIC</Microsoft.Help.F1>
      </entry>
   </children>
</entry>
```

Figure 16-5 shows a table of contents hierarchy in the Help viewer.

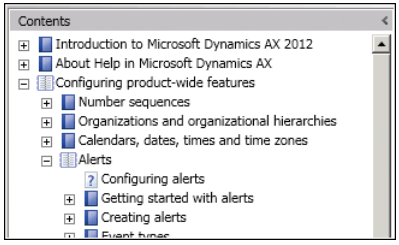


FIGURE 16-5 A table of contents hierarchy.

Create non-HTML content

Although the Help viewer cannot display a non-HTML file, it can open another application that can display the file; for example, it can use Word to open a .docx file.

To link to a non-HTML file from the Help viewer, you must have these components:

- You must include an HTML file that contains the metadata properties that the Help system requires. This file must have the same name as the non-HTML file.
- You must include a script that targets the non-HTML file. This file must be published to the same folder on the Help server as the file with the metadata properties.
- Computers with the Help viewer installed must have an application that can display the non-HTML file. The application must be the default application for that type of file.

The following sections describe how to create non-HTML content and the required metadata file.

Create the content

Open the application that you want to create the content with, and then create a new file. For example, open Word, and create a new document.

Tip If you use Word, use one of the templates included with the Help system to quickly create content that resembles existing Help content. For information about accessing the Word templates, see the section "Create content," earlier in this chapter.

Add content to the file—typically, a title, section headings, and paragraphs. Save the file, and note the file name. You will use this file name when you create the HTML file with the metadata.

Create the HTML metadata file

Create a new file, and add the HTML for a basic webpage, as in the following example:

```
<html>
   <head>
   </head>
</html>
```

Add a *<meta>* element to the file for each of the following metadata properties:

Property	Required	Description
description	No	A brief summary of the content. This appears in a list of search results.
documentSets	Yes	Set this property to "*UserDocumentation*".
Microsoft.Help.F1	Yes	A topic ID for the content. If the content applies to more than one topic, use a semicolon-delimited list of topic IDs.
Microsoft.Help.Id	Yes	A text value that uniquely identifies the content.
Microsoft.Help.Keywords	No	A semicolon-delimited list of keywords. Optional.
ms.locale	Yes	The language for the content. For example, use "*EN-US*" for U.S. English.
publisher	Yes	Your publisher ID. For more information, see the section "Publish content," later in this chapter.
suppressedPublishers	No	A semicolon-delimited list of publisher IDs. Hiding content produced by other publishers is optional.
Title	Yes	The text that is displayed in the title bar of the Help viewer.

The following example shows the metadata for the companion HTML file:

```
<head>
   <meta name="Title" content="Sample content" />
   <meta name="Microsoft.Help.Id" content="8D937F19-3A00-4F37-A316-0A48D052D627" />
   <meta name="ms.locale" content="En-Us" />
   <meta name="publisher" content="Microsoft" />
   <meta name="documentSets" content="UserDocumentation" />
   <meta name="Microsoft.Help.Keywords" content="" />
   <meta name="suppressedPublishers" content="" />
   <meta name="Microsoft.Help.F1" content="SampleContent" />
   <meta name="description" content="An example of non-HTML content that was published to the
Help system." />
</head>
```

Add a *<script>* element and specify the non-HTML file that you want to open. Specify the script type as *javascript* and use the *window.location* object to specify the file. The following HTML example shows a *<script>* element that opens the file *SampleContent.docx*:

```
<script type="text/javascript">
   <!--
      window.location="SampleContent.docx"
   //-->
</script>
```

When you save the file, the file name extension must be .htm.

Important The file name must match the file name of the non-HTML file. For example, use SampleContent.htm to match the SampleContent.docx example.

Publish content

To publish new or updated content and table of contents entries, you copy HTML or XML files to the Help server.

After creating or updating content, use these guidelines to ensure that you are ready to publish and to ensure that your content is visible on the Help server:

- Maintain a separate set of folders for each publisher (see Figure 16-6). This will ensure that content from a particular publisher doesn't get overwritten accidentally.
- Make sure that you have the correct permissions on the Help server to add files and folders.
- Add your publisher ID to the *Web.config* file of the Help server to determine where your documentation appears in the table of contents and in search results.
- If you have non-HTML content, you must include an HTML file with the required properties. The HTML file must be in the same folder as the non-HTML document. For more information, see the section "Create non-HTML content," earlier in this chapter.

Although this is not required, you can add subfolders to the content folder that specify your publisher ID and the language of your content, as shown in Figure 16-6. This can prevent files from other publishers from being overwritten—for example:

C:/inetpub/wwwroot/DynamicsAX6HelpServer/content/Contoso/EN-US

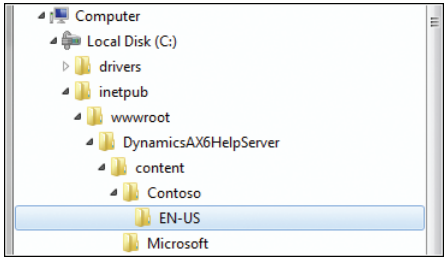


FIGURE 16-6 Help server folders for the publishers Contoso and Microsoft.

Details about the subfolders that you can add appear in the following table:

Folder Name	Recommended/ Optional	Example	Description
Publisher ID	Recommended	Contoso	Specify a unique name for the publisher—typically, your publisher ID from the metadata properties of your content. The publisher ID folder typically contains subfolders for the languages of your content.
Language	Recommended	EN-US	Specify the language of your content—typically, the language from the metadata properties of your content. You cannot change the value of the *Language* metadata property of content by changing the name of the folder where it is published.
Other	Optional	TOCResources	Add subfolders to help organize related content.

To delete existing content, remove the file that contains the content from the Help server.

Tip If you remove a topic, also update the topic from the table of contents and any cross-references.

Publication is completed when WSS adds metadata from your HTML or XML file to the search index. Make sure that WSS is running on the Help server.

The Help server uses the search service and its index to locate each topic that matches a Help request. You will not see newly published content in the Help viewer until that content is indexed.

Note WSS is a low-priority service that runs after higher-priority services. The time between publishing and viewing your content can vary. If you publish just a few files to a Help server that was previously indexed, the new files should be immediately indexed.

After WSS has indexed your files, use the Help viewer to view your new content. If you cannot see your content, check to ensure that the content has been indexed.

Add a publisher to the *Web.config* file

To refine search results, summary pages, and table of contents entries, the Help server keeps a list of publishers in the *Web.config* file. You update this list to complete the following tasks:

- Add or remove a publisher from the Search Options menu of the Help viewer (Figure 16-7) to restrict your search to content created by the specified publisher.

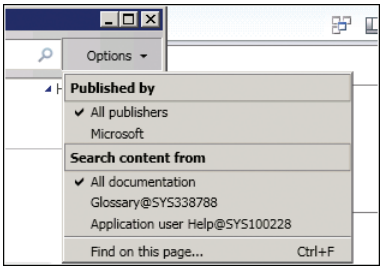


FIGURE 16-7 Publishers on the Search Options menu.

- List content from one publisher before or after content from another publisher. If your Help request includes content from more than one publisher, the summary page uses the publisher list to determine the sort order of the content.
- Specify where a group of entries in the table of contents appears. The Help server groups entries by publisher. When the Help server sends the table of contents to the Help viewer, the server uses the publisher list in the *Web.config* file to determine the order of the entries.

Before making changes to the *Web.config* file, save a copy of the file.

If you do not add your publisher ID to the *Web.config* file, the Help server determines the location of your content.

To add a publisher to the *Web.config* file:

1. Open the *Web.config* file in a text editor. To change the file, you might have to copy the file to a separate working folder. The *Web.config* file is located here:

   C:\inetpub\wwwroot\DynamicsAX6HelpServer

2. Add a publisher to the list. The list of publishers is in the *dynamicsHelpConfig* section.

   The following table specifies the required attributes of the <*publisher*> element:

Attribute Name	Value
publisherId	A value that uniquely identifies the publisher. The ID must match the publisher ID specified in the metadata for content elements and the table of contents.
name	The text to be displayed as the name of the publisher. The Search Options menu of the Help viewer displays this name.

Notice the order of the publishers in the following example. If a summary page includes content from both publishers, content from the first publisher is listed before content from the second:

```
<publishers>
   <add publisherId="Contoso" name="Contoso" />
   <add publisherId="Microsoft" name="Microsoft" />
</publishers>
```

3. Save your changes to the *Web.confi*g file. If the file is in a working folder, copy your updated *Web.config* file to the DynamicsAX6HelpServer folder on your Help server.

Publish content to the Help server

Before you add your content to the Help server, you can add subfolders to organize your files. Although not required, this can prevent files with similar names from other publishers from being overwritten. If you publish many files, you can add subfolders by subject to help organize your content.

Caution When you publish, be careful not to accidentally overwrite existing files that have the same file name. If you overwrite a file, you lose the Help documentation that was contained in the original file.

To add folders to the file system of the Help server:

1. In Windows Explorer, open the content folder on the Help server—typically, here:

   C:\inetpub\wwwroot\DynamicsAX6HelpServer\content

2. Add a publisher folder, using your publisher ID or name as the folder name.

3. Add language folders for the languages of your content. Name folders by using the same language code that is used in the language metadata of your content files—for example, "EN-US" for U.S. English.

To publish content:

1. In Windows Explorer, copy the files that you want to publish to the appropriate folders, such as

   C:\inetpub\wwwroot\DynamicsAX6HelpServer \content\<publisher ID>\<language>

   The following table summarizes the different files that accompany each type of content file:

Document Type	Description
HTML	Copy the .htm or .html files that you want to publish.
Word	Copy the .mht, .docm, or .docx files that you want to publish. Also copy the .htm files that contain the document properties for the Word files.
Other	Copy the document files that you want to publish. Also copy the .htm files that contain the document properties for the document files.

If you add many files, WSS takes several minutes to index all the files.

2. Open each content topic in the Help viewer and verify that your content was published.

To publish table of contents entries:

1. In Windows Explorer, copy your TableOfContents.xml file to the folder on the Help server that matches the publisher and language metadata in the XML file, such as

   C:\inetpub\wwwroot\DynamicsAX6HelpServer\content\<publisherID>\<language>\TOCResources

2. Open the Help viewer and verify that your content was published.

Set Help document set properties

A *document set* is a collection of content associated with a Microsoft Dynamics AX workspace—either the Microsoft Dynamics AX client or the Development Workspace. A workspace can be associated with only one document set. Typically, you use *UserDocumentation* as the document set for any content that you publish. If you add a new document set and associate it with a workspace, you will no longer see content from the document set that you replaced.

Document sets are located in an AOT node named *Help Document Sets*. Document sets have properties that help you manage the relationship between a workspace and a document set.

Property	Type	Description
DocumentSetName	String	The unique name of the document set. The name is limited to 40 characters and cannot contain spaces. Use the value of this property when you set the value of the *DocumentSets* metadata in a content file.
DocumentSetDescription	String	The text to display for the document set. This appears in the Search content from the list of the Options menu of the Help viewer.
AddToApplicationHelpMenu	Boolean	Set to *Yes* when you want the document set to appear on the Help menu in the Microsoft Dynamics AX client.
AddToDeveloperHelpMenu	Boolean	Set to *Yes* when you want the document set to appear on the Help menu of the Development Workspace.
UserDocumentSet	Boolean	Set to *Yes* when you want to associate the document set with the Microsoft Dynamics AX client. If you set this property to *No*, you will not be able to view the context-sensitive (F1) Help that was published by Microsoft.
DeveloperDocumentSet	Boolean	Set to *Yes* when you want to associate the document set with the Development Workspace. If you set this property to *No*, you will not be able to view the context-sensitive (F1) Help that was published by Microsoft.
ContentLocation	Enumeration	An enumeration value that specifies where to retrieve documentation: • Use an enumeration value of "1" and the label "Help server" with any document set that is published on the Help server. • Use an enumeration value of "2" and the label "World Wide Web" with any documentation that is stored on MSDN or a similar website. This option is required for the *DeveloperDocumentation* documentation set and should not be used with any other document set.

Troubleshoot the Help system

This section describes solutions to the two most common problems that might occur when customizing the Help system.

The Help viewer cannot display content

If the Help viewer cannot display content, check the following possible solutions.

Help server

If you use more than one Help server for development, testing, and production, make sure that the Help viewer connects to the server where you published your changes. To view the URL of the Help service in Microsoft Dynamics AX, click Administration > Setup > Help System Parameters.

Make sure that the web service and application pool for the Help service are running. Click Start > All Programs > Administrative Tools > Internet Information Services (IIS) Manager.

WSS

Content does not appear in the Help viewer until it has been indexed by WSS. Right-click the taskbar, click Start Task Manager, click the Services tab, and then, in the Name column, find Wsearch, and verify that "Running" appears in the Status column. If WSS is running, you might have to give indexing more time to find your content. Indexing slows or stops when the server is busy.

If WSS is not running, right-click Wsearch, and then click Start Service. Allow WSS to find and index the files that you published.

Check whether the Help server or WSS logged any error messages in the application log of the server. In Event Viewer, click Application Log.

Content

Check whether the Help server can open and process the HTML file of your content. If the Help server cannot locate the file, or the file does not include the required metadata, the Help server does not send the content to the Help viewer.

- Make sure that the HTML file is in the correct folder on the Help server.
- Make sure that the content file includes all required metadata with the correct syntax.
- Make sure that there are no errors in the XHTML of your content files.

The Help viewer cannot display the table of contents

Check whether the Help server can open and process the XML file for the table of contents entries. If the Help server cannot locate the file, or the file does not include the required metadata, the Help server will not add your entries to the table of contents.

- Make sure that the XML file is in the correct folder.
- Make sure that the file includes all required metadata in the correct syntax.
- Make sure that the ID in the *Microsoft.Help.F1* property of the table of contents entry identifies only a single topic.

PART III

Under the hood

CHAPTER 17 The database layer 575

CHAPTER 18 Batch framework 613

CHAPTER 19 Application frameworks 633

CHAPTER 20 Reflection 669

CHAPTER 21 Application models 687

CHAPTER 17

The database layer

In this chapter

Introduction . . . 577
Temporary tables . . . 577
Surrogate keys . . . 585
Alternate keys . . . 587
Table relations . . . 588
Table inheritance . . . 594
Unit of Work . . . 599
Date-effective framework . . . 601
Full-text support . . . 606
The QueryFilter API . . . 607
Data partitions . . . 610

Introduction

The Microsoft Dynamics AX 2012 application run time provides robust database features that make creating an enterprise resource planning (ERP) application much easier. Many new and powerful database features have been added to Microsoft Dynamics AX 2012. This chapter focuses on several of these new capabilities. The information provided here introduces the features, provides information about how to use them in an application, and, when appropriate, explains in detail how each feature works.

Many database features, such as optimistic concurrency control (OCC), transaction support, and the query system, have been available in Microsoft Dynamics AX for several releases. For detailed information about these and other database features in Microsoft Dynamics AX, see the "Database" section in the Microsoft Dynamics AX 2012 software development kit (SDK) at *http://msdn.microsoft.com/en-us/library/aa588039.aspx*.

You can also refer to previous editions of "*Inside Microsoft Dynamics AX*," which contain useful information about database functionality that still applies to Microsoft Dynamics AX 2012.

Temporary tables

By default, any table that is defined in the Application Object Tree (AOT) is mapped in a one-to-one relationship to a permanent table in the underlying relational database. Microsoft Dynamics AX also supports the functionality of temporary tables. In previous releases, Microsoft Dynamics AX provided

the capability to create *InMemory* temporary tables that are mapped to an indexed sequential access method (ISAM) file-based table that is available only during the run-time scope of the Application Object Server (AOS) or a client. Microsoft Dynamics AX 2012 provides a new type of temporary table that is stored in the TempDB database in Microsoft SQL Server.

InMemory temporary tables

The ISAM file that represents an *InMemory* temporary table contains the data and all of the indexes that are defined for the table in the AOT. Because working on smaller datasets is generally faster than working on larger datasets, the Microsoft Dynamics AX run time monitors the size of each *InMemory* temporary table. If the size is less than 128 kilobytes (KB), the temporary table remains in memory. If the size exceeds 128 KB, the temporary table is written to a physical ISAM file. Switching from memory to a physical file affects performance significantly. A file with the naming syntax *$tmp<8 digits>.$$$* is created when data is switched from memory to a physical file. You can monitor the threshold limit by noting when this file is created.

Although *InMemory* temporary tables don't map to a relational database, all of the data manipulation language (DML) statements in X++ are valid for tables that operate as *InMemory* temporary tables. However, the Microsoft Dynamics AX run time executes some of the statements in a downgraded fashion because the ISAM file functionality doesn't offer the same functionality as a relational database. For example, set-based operators always execute as record-by-record operations.

Using *InMemory* temporary tables

When you declare a record buffer for an *InMemory* temporary table, the table doesn't contain any records. You must insert records to work with the table. The *InMemory* temporary table and all of the records are lost when no declared record buffers point to the temporary dataset.

Memory and file space aren't allocated to the *InMemory* temporary table until the first record is inserted. The temporary table is located on the tier where the first record was inserted. For example, if the first insert occurs on the server tier, the memory is allocated on this tier, and eventually the temporary file will be created on the server tier.

Important Use temporary tables carefully to ensure that they don't cause excessive round trips between the client and the server, resulting in degraded performance. For more information, see Chapter 13, "Performance."

A declared temporary record buffer contains a pointer to the dataset. If you use two temporary record buffers, they point to different datasets by default, even though the table is of the same type. To illustrate this, the X++ code in the following example uses the TmpLedgerTable temporary table defined in Microsoft Dynamics AX 2012. The table contains four fields: *AccountName*, *AccountNum*, *CompanyId*, and *LedgerDimension*. The *AccountNum* and *CompanyId* fields are both part of a unique index, *AccountNumIdx*, as shown in Figure 17-1.

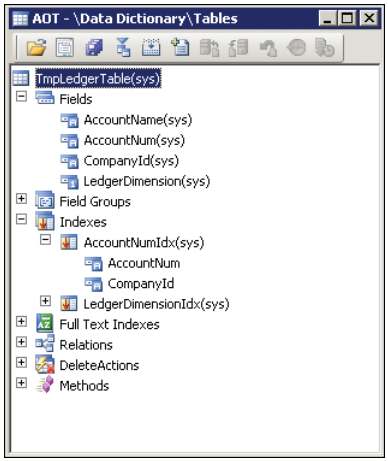


FIGURE 17-1 TmpLedgerTable temporary table.

The following X++ code shows how the same record can be inserted in two record buffers of the same type. Because the record buffers point to two different datasets, a "duplicate value in index" failure doesn't result, as it would if both record buffers pointed to the same temporary dataset, or if the record buffers were mapped to a database table.

```
static void TmpLedgerTable(Args _args)
{
    TmpLedgerTable tmpLedgerTable1;
    TmpLedgerTable tmpLedgerTable2;

    tmpLedgerTable1.CompanyId = 'dat';
    tmpledgerTable1.AccountNum = '1000';
    tmpLedgerTable1.AccountName = 'Name';
    tmpLedgerTable1.insert(); // Insert into tmpLedgerTable1's dataset.

    tmpLedgerTable2.CompanyId = 'dat';
    tmpledgerTable2.AccountNum = '1000';
    tmpLedgerTable2.AccountName = 'Name';
    tmpLedgerTable2.insert(); // Insert into tmpLedgerTable2's dataset.
}
```

To have the record buffers use the same temporary dataset, you must call the *setTmpData* method on the record buffer, as illustrated in the following X++ code. In this example, the *setTmpData* method is called on the second record buffer and is passed in the first record buffer as a parameter.

```
static void TmpLedgerTable(Args _args)
{
    TmpLedgerTable tmpLedgerTable1;
    TmpLedgerTable tmpLedgerTable2;

    tmpLedgerTable2.setTmpData(tmpLedgerTable1);

    tmpLedgerTable1.CompanyId = 'dat';
```

```
    tmpledgerTable1.AccountNum = '1000';
    tmpLedgerTable1.AccountName = 'Name';
    tmpLedgerTable1.insert(); // Insert into shared dataset.

    tmpLedgerTable2.CompanyId = 'dat';
    tmpledgerTable2.AccountNum = '1000';
    tmpLedgerTable2.AccountName = 'Name';
    tmpLedgerTable2.insert(); // Insert will fail with duplicate value.
}
```

The preceding X++ code fails on the second insert operation with a "duplicate value in index" error because both record buffers point to the same dataset. You would notice similar behavior if, instead of calling *setTmpData*, you simply assigned the second record buffer to the first record buffer, as illustrated here:

```
tmpLedgerTable2 = tmpLedgerTable1;
```

However, the variables would point to the same object, which means that they would use the same dataset.

When you want to use the *data* method to copy data from one temporary record buffer to another, where both buffers point to the same dataset, write the code for the copy operation as follows:

```
tmpLedgerTable2.data(tmpLedgerTable1);
```

Warning The connection between the two record buffers and the dataset is lost if the code is written as *tmpLedgerTable2 = tmpLedgerTable1.data*. In this case, the temporary record buffer points to a new record buffer that has a connection to a different dataset.

As mentioned earlier, if no record buffer points to the dataset, the records in the temporary table are lost, the allocated memory is freed, and the physical file is deleted. The following X++ code example illustrates this situation, in which the same record is inserted twice using the same record buffer. But because the record buffer is set to *null* between the two insert operations, the first dataset is lost, so the second insert operation doesn't result in a duplicate value in the index because the new record is inserted into a new dataset.

```
static void TmpLedgerTable(Args _args)
{
    TmpLedgerTable tmpLedgerTable;

    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // Insert into first dataset.

    tmpLedgerTable = null; // Allocated memory is freed
                           // and file is deleted.
    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
```

```
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // Insert into new dataset.
}
```

Notice that none of these *InMemory* temporary table examples uses the *ttsbegin*, *ttscommit*, and *ttsabort* statements. These statements affect only ordinary tables that are stored in a relational database. For example, the following X++ code adds data to an *InMemory* temporary table. Because the table is an *InMemory* temporary table, the value of the *accountNum* field is printed to the Infolog even though the *ttsabort* statement executes.

```
static void TmpLedgerTableAbort(Args _args)
{
    TmpLedgerTable tmpLedgerTable;

    ttsbegin;
    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // Insert into table.
    ttsabort;

    while select tmpLedgerTable
    {
        info(tmpLedgerTable.AccountNum);
    }
}
```

To cancel the insert operations on the table in the preceding scenario successfully, you must call the *ttsbegin* and *ttsabort* methods on the temporary record buffer instead, as shown in the following example:

```
static void TmpLedgerTableAbort(Args _args)
{
    TmpLedgerTable tmpLedgerTable;

    tmpLedgerTable.ttsbegin();
    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // Insert into table.
    tmpLedgerTable.ttsabort();

    while select tmpLedgerTable
    {
        info(tmpLedgerTable.AccountNum);
    }
}
```

When you work with multiple temporary record buffers, you must call the *ttsbegin*, *ttscommit*, and *ttsabort* methods on each record buffer because there is no correlation between the individual temporary datasets.

Considerations for working with *InMemory* temporary tables

When working with *InMemory* temporary tables, keep the following points in mind:

- When exceptions are thrown and caught outside the transaction scope, if the Microsoft Dynamics AX run time has already called the *ttsabort* statement, temporary data isn't rolled back. When you work with temporary datasets, make sure that you're aware of how the datasets are used both inside and outside the transaction scope.
- The database-triggering methods on temporary tables behave almost the same way as they do with ordinary tables, but with a few exceptions. When *insert*, *update*, and *delete* are called on the temporary record buffer, they don't call any of the database-logging or event-raising methods on the application class if database logging or alerts have been set up for the table.

Note In general, you can't set up logging or events on *InMemory* temporary tables that you define. However, because ordinary tables can be changed to temporary tables, logging or events might already be set up.

- Delete actions are also not executed on *InMemory* temporary tables. Although you can set up delete actions, the Microsoft Dynamics AX application run time doesn't try to execute them.
- Microsoft Dynamics AX lets you trace Transact-SQL statements, either from within the Microsoft Dynamics AX Windows client, or from the Microsoft Dynamics AX Configuration Utility or the Microsoft Dynamics AX Server Configuration Utility. However, Transact-SQL statements can be traced only if they are sent to the relational database. You can't trace data manipulation in *InMemory* temporary tables with these tools. However, you can use the Microsoft Dynamics AX Trace Parser to accomplish this. For more information, see the section, "Microsoft Dynamics AX Trace Parser" in Chapter 13.
- You can query a record buffer to find out whether it is acting on a temporary dataset by calling the *isTmp* record buffer method, which returns a value of *true* or *false* depending on whether the table is temporary.

TempDB temporary tables

The application code in Microsoft Dynamics AX often uses temporary tables for intermediate storage. This requires performing joins with regular tables and in some cases, executing set-based operations. But *InMemory* temporary tables provide restricted support for joins. These joins are performed by the data layer in the AOS, which does not give ideal performance. Also, as mentioned earlier, set-based operations are always downgraded to row-by-row operations for *InMemory* tables. *TempDB* temporary tables have been added to Microsoft Dynamics AX to provide a high-performance solution for these scenarios. Because these temporary tables are stored in the SQL Server database, database operations such as joins can be used.

TempDB temporary tables use the same X++ programming constructs as *InMemory* temporary tables. The key difference is that they are stored in the SQL Server *TempDB* database. The following code example shows the usage of a *TempDB* temporary table:

```
void select2Instances()
{
    TmpDBTable1 dbTmp1;
    TmpDBTable1 dbTmp2;

    dbTmp1.Field1 = 1;
    dbTmp1.Field2 = 'First';
    dbTmp1.insert();

    dbTmp2.Field1 = 2;
    dbTmp2.Field2 = 'Second';
    dbTmp2.insert();
    info("First Instance.");
    while select * from dbTmp1
    {
        info(strfmt("%1 - %2", dbTmp1.Field1, dbTmp1.Field2));
    }
    info("Second Instance.");
    while select * from dbTmp1
    {
        info(strfmt("%1 - %2", dbTmp2.Field1, dbTmp2.Field2));
    }
}
```

This example uses a table called TmpDBTable1 that contains two fields. The *TableType* property for the TmpDBTable1 table is set to *TempDB*. Similar to an *InMemory* temporary table, the *TempDB* temporary table is created only when the data is inserted into the table buffer. To see the data in the temporary table, insert a breakpoint before the first *select* statement in the X++ code, and then open SQL Server Management Studio (SSMS) to examine the tables that are created in the *TempDB* system database. Each table buffer instance for the temporary table has a corresponding table in the database in the following format: *t<table_id>_GUID*. In the example, the table ID of the TmpDBTable1 table is *101420*. This means that two tables were created in the *TempDB* database, with one of them having the table name t101420_E448847EACA4482997F4CD8BCAAAE0CE. After the method runs and the table buffers are destroyed, these two tables are truncated. The Microsoft Dynamics AX run time uses a pool to keep track of these tables in the *TempDB* database. The run time will reuse one of these table instances when a TmpDBTable1 table buffer is created again in X++.

Creating temporary tables

You can create temporary tables in the following ways:

- At design time by setting metadata properties.
- At configuration time by enabling licensed modules or configurations.
- At application run time by writing explicit X++ code.

The following sections describe each method.

Design time

To define a table as temporary, you must set the appropriate value in the *TableType* property for the table resource. By default, the *TableType* property is set to *Regular*. To create a temporary table, choose one of the other two options: *InMemory* or *TempDB*, as shown in Figure 17-2. The temporary tables are created in memory, and are backed by a file or created in the *TempDB* database when needed.

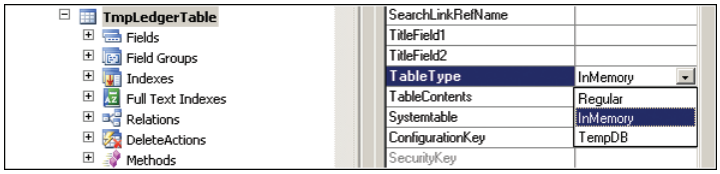


FIGURE 17-2 Marking a table as temporary at design time.

Tip Tables that you define as temporary at design time should have *Tmp* inserted as part of the table name instead of at the beginning or end of the name; for example, InventCostTmpTransBreakdown. This improves readability of the X++ code when temporary tables are explicitly used. In previous versions of Microsoft Dynamics AX, the best practice was to prefix temporary tables with *Tmp*, which is why a number of temporary tables still use this convention.

Configuration time

When you define a table by using the AOT, you can attach a configuration key to the table by setting the *ConfigurationKey* property on the table. The property belongs to the Data category of the table properties.

When the Microsoft Dynamics AX run time synchronizes the tables with the database, it synchronizes tables for all modules and configurations, regardless of whether they're enabled except the SysDeletedObjects configuration keys. Whether a table belongs to a licensed module or an enabled configuration depends on the settings in the *ConfigurationKey* property. If the configuration key is disabled, the table is disabled and behaves like a *TempDB* temporary table. Therefore, no run-time error occurs when the Microsoft Dynamics AX run time interprets X++ code that accesses tables that aren't enabled. For more information about the SysDeletedObjects configuration key, see "Best Practices: Tables" at *http://msdn.microsoft.com/en-us/library/aa876262.aspx*.

Note Whether it is enabled doesn't affect a table that is already defined as a temporary table. The table remains temporary even though its configuration key is disabled, and you can expect the same behavior regardless of the configuration key setting.

Run time

You can use X++ code to turn an ordinary table into an *InMemory* temporary table by calling the *setTmp* method on the record buffer. From that point forward, the record buffer is treated as though the *TableType* property on the table is set to *InMemory*.

Note You can't define a record buffer of a temporary table type and turn it into an ordinary table, partly because there is no underlying table in the relational database.

The following X++ code illustrates the use of the *setTmp* method, in which two record buffers of the same type are defined; one is temporary, and all records from the database are inserted into the temporary version of the table. Therefore, the temporary record buffer points to a dataset that contains a complete copy of all of the records from the database belonging to the current company.

```
static void TmpCustTable(Args _args)
{
    CustTable custTable;
    CustTable custTableTmp;

    custTableTmp.setTmp();
    ttsbegin;
    while select custTable
    {
        custTableTmp.data(custTable);
        custTableTmp.doInsert();
    }
    ttscommit;
}
```

Notice that the preceding X++ code uses the *doInsert* method to insert records into the temporary table. This prevents execution of the overridden *insert* method. The *insert* method inserts records in other tables that aren't switched automatically to temporary mode just because the *custTable* record buffer is temporary.

Caution Use great care when changing an ordinary record buffer to a temporary record buffer because application logic in overridden methods that manipulates data in ordinary tables could execute inadvertently. This can happen if the temporary record buffer is used in a form and the form application run time calls the database-triggering methods.

Surrogate keys

The introduction of surrogate keys is a significant change to table keys in Microsoft Dynamics AX 2012. A surrogate key is a system-generated, single-column primary key that does not carry domain-specific semantics. It is specific to the Microsoft Dynamics AX installation that generates the key. A surrogate key value cannot be generated in one installation and used in another installation.

The natural choice for a surrogate key in Microsoft Dynamics AX is the *RecId* column. However, in Microsoft Dynamics AX 2009 and earlier, the *RecId* index is still paired with the *DataAreaId* column for company-specific tables. In Microsoft Dynamics AX 2012, the *RecId* index does not contain the *DataAreaId* column and becomes a single-column unique key.

Surrogate key support is enabled when you create a new table the AOT. To use the surrogate key pattern for a table, set the *PrimayIndex* property to *SurrogateKey*, as shown in Figure 17-3.

Note The *SurrogateKey* value is not available for tables that were created in an earlier version of Microsoft Dynamics AX. If you want to implement a surrogate key for an existing table, you must re-create the table in the AOT.

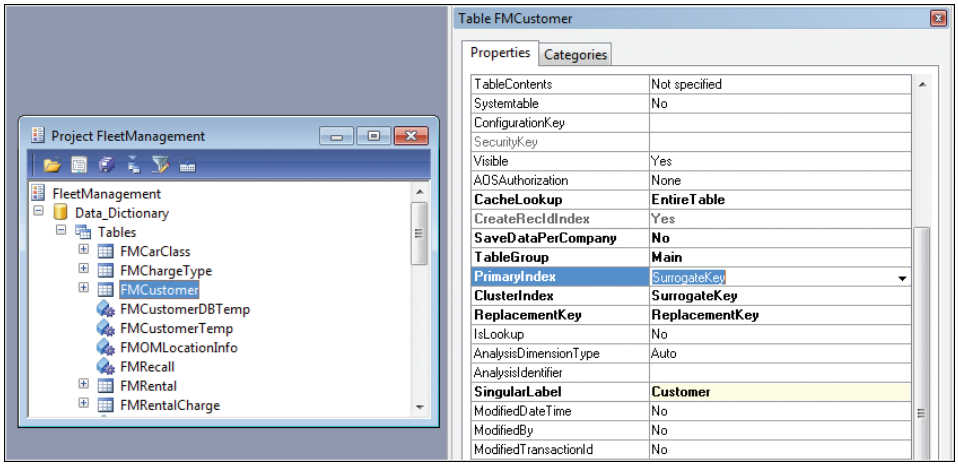


FIGURE 17-3 Surrogate key in the FMCustomer table.

Defining a surrogate key on a table in Microsoft Dynamics AX has several benefits. The first benefit is performance. When a surrogate key and the foreign key that references it are used to join two tables, performance is improved when compared to joins created with other data types. The benefit is more prominent when compared to Microsoft Dynamics AX 2009 and earlier versions because the kernel automatically adds the *DataAreaId* column to any key defined for any company-specific table. You identify these company-specific tables through the *SaveDataPerCompany* property. Without a surrogate key, the join must be based on at least on two columns, with one of them being the *DataAreaId* column.

The second benefit of using a surrogate key is that a surrogate key value never changes, which eliminates the need to change the values of foreign keys. For example, the Currency table (Figure 17-4) uses the *CurencyCodeIdx* index as the primary index, which contains the *CurrencyCode* column. The Ledger table has two foreign keys in the Currency table that are based on the *CurrencyCode* column. If there is ever a need to change the *CurrencyCode* value for a record in the Currency table, the corresponding records in the Ledger table also must be updated. But if a surrogate key were used for the Currency table and the Ledger table holds the surrogate foreign key, you could update the

CurrencyCode value in the Currency table without affecting the key relationship between the row in the Currency table and the rows in the Ledger table.

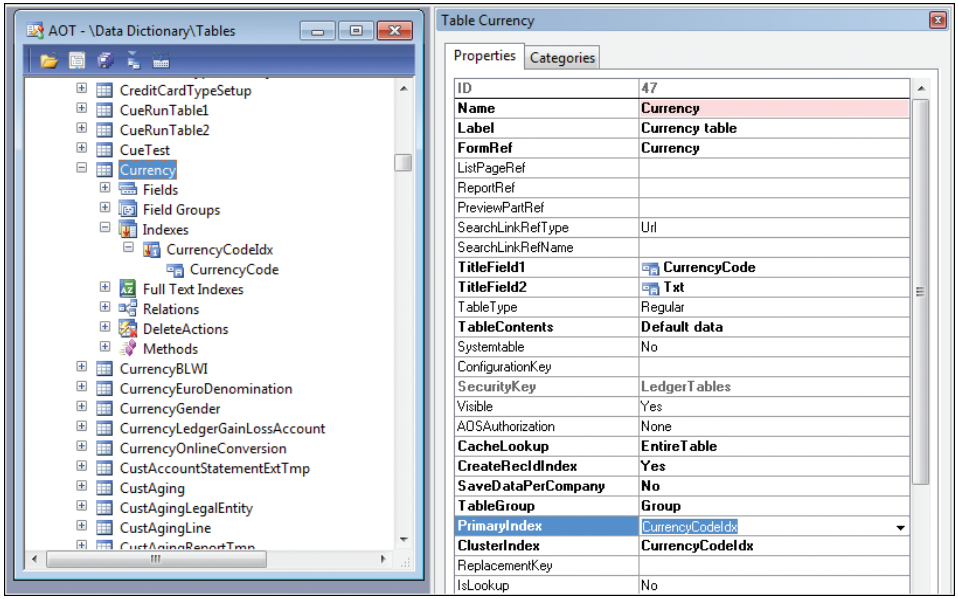


FIGURE 17-4 Currency table without a surrogate key.

The third benefit of using a surrogate key is that it is always a single-column key. Some features of SQL Server, such as full-text search, require a single-column key. Using a surrogate key lets you take advantage of these features.

Surrogate keys do have some drawbacks. The most prominent drawback is that the key value is not human-readable. When you look at the foreign keys on a related table, it is not easy to determine what the related row is. To display meaningful information that identifies the foreign key, some human-readable information from the related entity must be retrieved and displayed instead. This requires a join to the related table, and the join adds performance overhead.

Alternate keys

For a table, a *candidate key* is a minimal set of columns that uniquely identifies each row in the table. An *alternate key* is a candidate key for a table that is not a primary key. In Microsoft Dynamics AX 2012, you can mark a unique index to be an alternate key.

Because Microsoft Dynamics AX already has the concept of a unique index, you might wonder what the value is of having the concept of an alternate key. Developers create unique indexes for various reasons. Typically, one unique index serves as the primary key. Sometimes, additional unique indexes are created for performance reasons, such as to support a specific query pattern. When you look at the unique indexes for a table, it is not always obvious which index is used for the primary key

and which have been added for other reasons. For example, if you were to extract your data model and present it to a business analyst, you would not want the analyst to see the keys that were created solely for performance reasons. You need a way to separate your semantic model from your physical data model for this purpose. Being able to designate the additional unique indexes as alternate keys helps you to achieve this.

Figure 17-5 shows a unique index that was added to the FMVehicleModel table of the Fleet Management sample application. This is not the primary index for the table, so the *AlternateKey* property for the index is set to *Yes*.

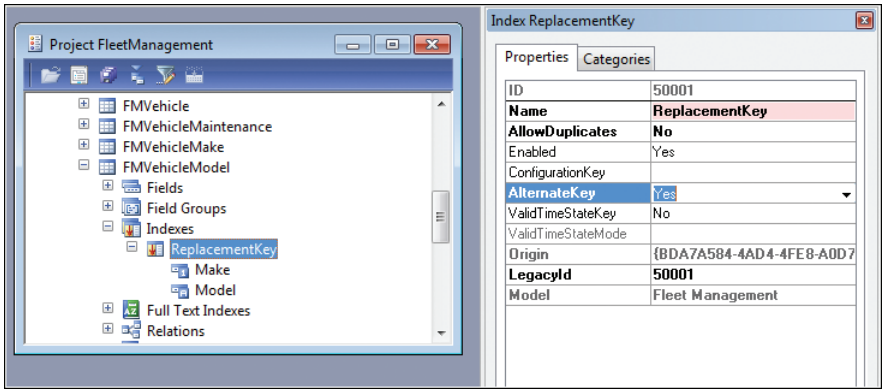


FIGURE 17-5 Alternate key on the FMVehicleModel table.

Table relations

Relationships between tables (called *relations* in Microsoft Dynamics AX) are key to the data model and run-time behavior. They are used in the following run-time scenarios in Microsoft Dynamics AX:

- Join conditions in the query or form data source and form dynalink
- Delete actions on tables
- Lookup conditions for bound (backed by a data source) or unbound controls on forms

Several changes and enhancements have been made for table relations in Microsoft Dynamics AX 2012.

EDT relations and table relations

In Microsoft Dynamics AX, table relations can be explicitly defined on tables, or they can be derived from relation properties that are defined on extended data types (EDTs) associated with table fields. The Microsoft Dynamics AX kernel looks up the relation properties for EDTs, and then for table relations. The order may be switched, depending on the scenario.

There are several issues with defining the relation properties on EDTs and mixing them with the relations defined on tables. First, EDT relations capture relations on only a single field. They cannot be used to capture multiple-field relations, which leads to incomplete relationships that are used in join or delete actions. Second, most of the properties for the relation depend on the context in which the relation is used. This context cannot be captured for EDT relations because they are stored in a central location in the AOT. For example, the role and related role name and cardinality and related cardinalities could be different for relations on different tables. Third, it is difficult to figure out how many relations are actually available for a table because the table relations give you only a partial view. You need to also look at relations that are defined on EDTs for the fields in the table and their base EDTs.

To address these issues, Microsoft Dynamics AX 2012 has migrated most of the EDT relations to table relations. To begin deprecating the *Relations* node under individual EDTs, the addition of new relations is not allowed. If an EDT has no nodes defined under the *Relations* node, the node is not displayed in Microsoft Dynamics AX 2012. An EDT has a new *Table References* node for cases where a control is bound directly to an EDT, and not through a table field. To reduce the work of manually adding a table relation that can be used in place of the EDT relation, you are prompted to create the table relation automatically when an EDT with a valid foreign key reference is used on a table field.

Because the Microsoft Dynamics AX run time performs lookups for EDT relations and table relations in a specific order, you need to ensure that the same relation gets picked up before and after the migration in all scenarios. This is especially important when migrating EDT relations in multiple table relations between two tables. The Microsoft Dynamics AX run time achieves backward compatibility by examining some properties that were set on table fields and table relations. These properties enable the run time to determine whether a table relation was created from an EDT relation that existed before. These properties are explained in Table 17-1.

TABLE 17-1 Properties of table fields.

Property location	Property name	Values	Description
Table relation	EDTRelation	Yes/No	Indicates to the kernel whether the relation was created because an EDT relation was migrated. The kernel uses this information to order and pick the relation for join, lookup, and delete actions when no explicit relation is specified.
Table field	IgnoreEDTRelation	Yes/No	Indicates whether the EDT relation on the field should be migrated to a table relation. The Microsoft Dynamics AX run time does not use this property; however, the EDT Relation Migration tool does.
Table relation link	SourceEDT	EDT name	Used by the Microsoft Dynamics AX run time to determine the cases in which the EDT relation is used.

You do not have to migrate an EDT relation manually to a table relation and then set the properties described in Table 17-1. The EDT Relation Migration tool can help with this process. You can access this tool from Tools > Code Upgrade > EDT Relation Migration Tool, as shown in Figure 17-6. For information about how to use this tool, see the topic "EDT Relation Migration Tool" at *http://msdn.microsoft.com/en-us/library/gg989788.aspx*.

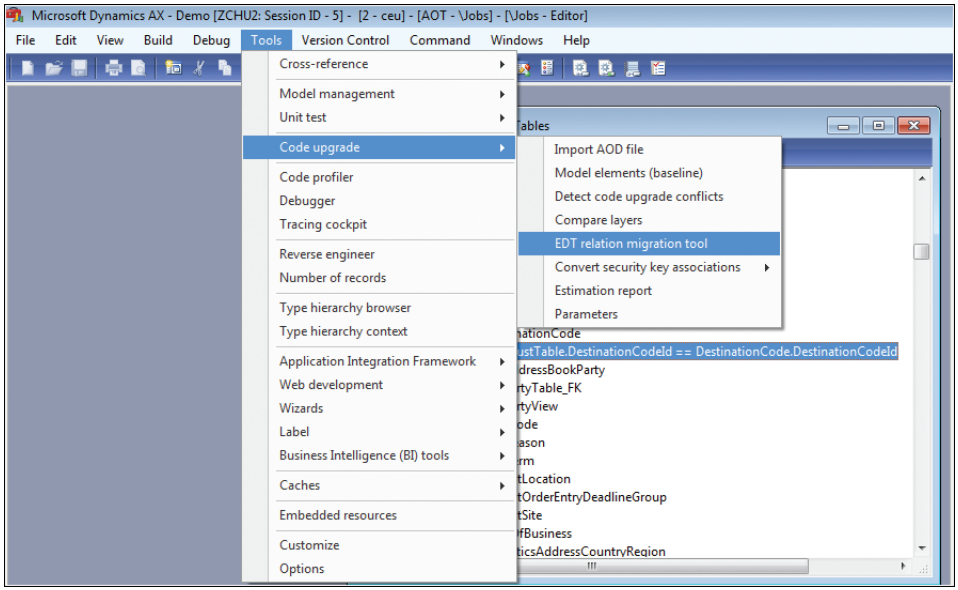


FIGURE 17-6 EDT Relation Migration tool.

After you migrate the EDT relations to table relations, verify that delete actions, query joins, and lookups work the same way they did before the migration. Focus on cases in which the migration of an EDT relation resulted in multiple table relations between two tables. The EDT Relation Migration tool lists the artifacts that are affected.

The following are some examples of the artifacts that might be affected. For example, the SalesTable table and the CustTable table have two relations between them defined on SalesTable. One of them was migrated from an EDT relation because it did not exist as a table relation before. The two relations are based on the fields *CustAccount* and *InvoiceAccount*. For delete actions, the relation based on *CustAccount* should be picked up before and after the migration. The *SalesHeading* query has a join between the SalesTable table and the CustTable table with the *Relations* property set to *Yes* on the data source for the SalesTable table. This query should pick up the relation based on *CustAccount* instead of the relation based on *InvoiceAccount* before and after the migration.

Foreign key relations

A goal for Microsoft Dynamics AX 2012 was to make the data model more consistent. This includes using surrogate key patterns when appropriate. It also includes using only primary keys as your foreign key references whenever possible. To facilitate the latter, Microsoft Dynamics 2012 introduces a special type of foreign key relation that you can use when creating relations between tables, as shown in Figure 17-7. A foreign key relation allows only two kinds of references to the related table. The first is a reference to the primary key. The second is a reference to a single-column alternate key of the related table. This reference to the single-column alternate key is provided to reduce the number of surrogate foreign key joins that were discussed in the "Surrogate keys" section, earlier in this chapter.

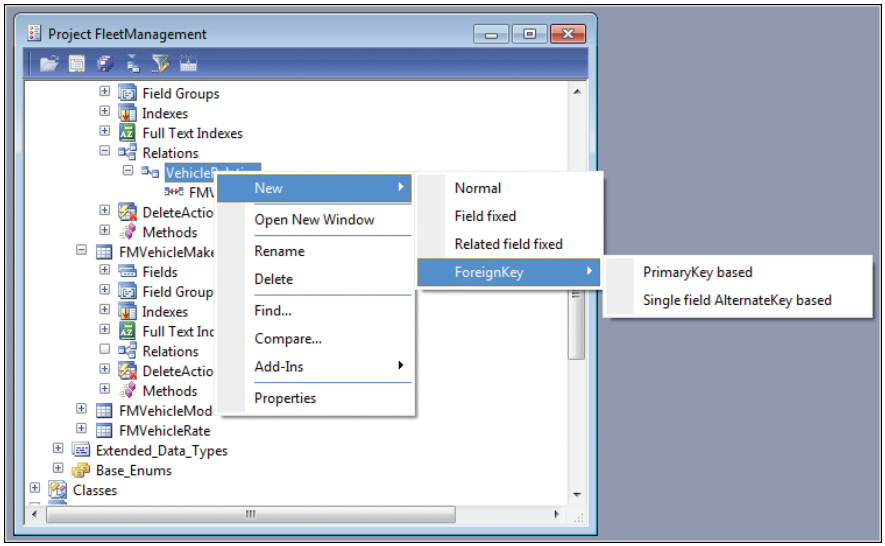


FIGURE 17-7 The new foreign key relation in Microsoft Dynamics AX 2012.

If you use a human-readable alternate key as your foreign key, you can display the foreign key on forms without the need for a join. You might wonder why only single-column alternate keys are allowed for foreign key references. This is to balance performance for a different usage pattern, when you actually want to join between the two tables (not for purposes of the user interface). Joins that are based on smaller columns (both the size of the columns and the number of columns) perform faster. By restricting the pattern to only single-column alternate keys, the performance degradation of the join is limited.

A consistent table relation pattern can result in performance benefits, too. For example, if one table references the CustTable table by using *keyA*, and another table references the CustTable table by using *keyB*, both tables must be joined to the CustTable table to correlate the rows in these two tables. However, if both of them use the same key, they can correlate directly, eliminating the need for joins.

Foreign key relations have some capabilities that other relations do not. For example, you can use them as join conditions in queries. This saves you from having to manually enter the field join conditions, which can be prone to error. Navigation property methods can also be generated for foreign key relations, as discussed in the next section.

The *CreateNavigationPropertyMethods* property

When you expand the *Relations* node for a table defined in the AOT, you can see the table relationships that are defined. The *CreateNavigationPropertyMethods* property, which is available only for foreign key relations, has special significance. Setting this property to *Yes,* as shown in Figure 17-8, causes kernel-generated methods on a table buffer to be created. You can use these methods to set, retrieve, and navigate to the related table buffer through the relation specified. The examples later in this section show the method signatures and usage patterns.

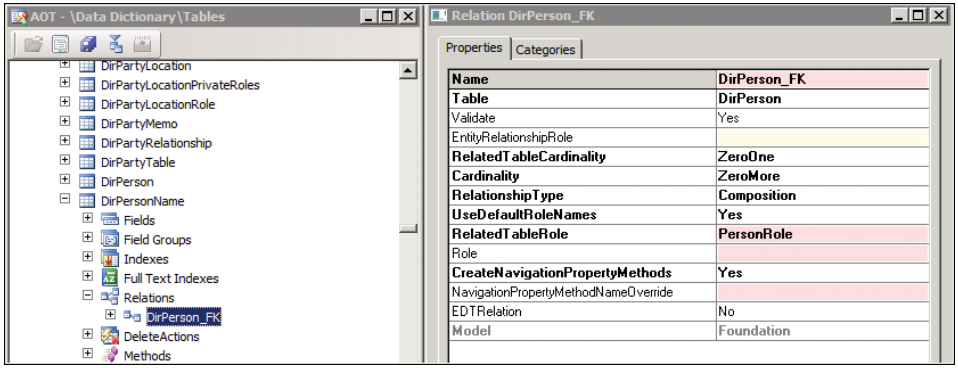


FIGURE 17-8 The *CreateNavigationPropertyMethods* property on a foreign key relation.

The navigation *setter* method links two related table buffers together. It is frequently used with the *UnitOfWork* class to create rows in the database from those table buffers. This effectively allows you to create an in-memory object graph with a related table buffer so that you can push them into the database with the proper relationship established among the rows. For more information, see the "Unit of Work" section later in this chapter.

The navigation *getter* method retrieves the related table buffer if a *setter* method has set it. Otherwise, the method retrieves the related table buffer from the database. This can effectively replace the *find* method pattern that is commonly used on tables. In the latter case, the table buffer that is returned is not linked to the table buffer on which the method was called. This means that the method will try to retrieve data from the database again. Note that when the navigation property getter method queries the database to get the related record, it selects all fields for that record. This can affect performance, particularly in cases where you had selected a smaller field list to achieve a performance benefit.

The following code uses the *DirPartyTable_FK* method to retrieve the related DirPartyTable table record for a customer with an account number of *1101* and prints the customer's name to the Infolog:

```
static void NavigationPropertyMethod(Args _args)
{
    CustTable cust;

    select cust where cust.AccountNum == '1101';

    // The DirPartyTable_FK() methods retrieves the related DirPartyTable record
    // through the DirPartyTable_FK role defined on the CustTable

    info(strFmt('Customer name for %1 is %2',cust.AccountNum, cust.DirPartyTable_FK().Name));
}
```

Figure 17-9 shows the output of this example in the Infolog.

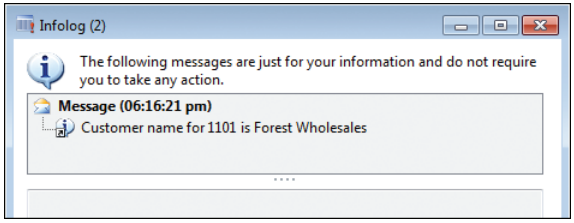


FIGURE 17-9 Output from the *DirPartyTable_FK* method example.

However, if the navigation property *setter* method is used to set the related DirPartyTable record, that record is always returned and the run time does not query the database:

```
static void NavigationPropertyMethodSetter(Args _args)
{
    CustTable cust;
    DirPartyTable dp;

    select cust where cust.AccountNum == '1101';
    dp.Name = 'NotARealCustomer';

    // Set the related DirPartyTable record

    cust.DirPartyTable_FK(dp);

    // The DirPartyTable_FK() methods retrieves the DirPartyTable record set above and
    // does not retrieve from the database.

    info(strFmt('Customer name for %1 is %2',cust.AccountNum, cust.DirPartyTable_FK().Name));
}
```

Figure 17-10 shows the output from this example in the Infolog.

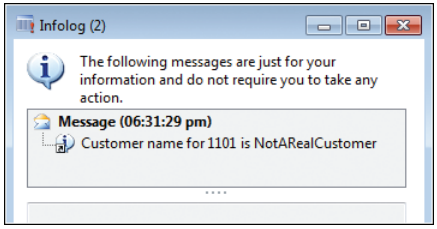


FIGURE 17-10 Output from an example with a navigation property *setter* method.

Each navigation method must have a name. Like any other method on the table, its name cannot conflict with other methods. By default, the *RelatedTableRole* property is used for the method name. An error is thrown during table compilation if a conflict with another method name is detected. If a conflict occurs, use the *NavigationPropertyMethodNameOverride* property to specify the name to use.

Table inheritance

Table inheritance has long been part of extended entity relationship (ER) modeling, but there was no built-in support for this in earlier versions of Microsoft Dynamics AX. Any inheritance or object-oriented characteristics had to be implemented manually by the developer. Microsoft Dynamics AX 2012 supports table inheritance natively from end to end, including modeling, language, run time, and user interface.

Modeling table inheritance

To model table inheritance in Microsoft Dynamics AX 2012, you must first create a root table and then create a derived table. These tasks are described in the following sections. Later sections describe how to work with existing tables, view the type hierarchy, and specify table behavior.

Create the root table

First, you must create the table that is the root of the table hierarchy. Before you create any fields for the table, set the *SupportInheritance* property to *Yes*. For the root table, you must add an *Int64* column that is named InstanceRelationType, which holds the information about the actual type of a specific row. This column should have the *ExtendedDataType* property set to *RelationType*, and the *Visible* property set to *No*. After you create this field, you must set the *InstanceRelationType* property for the base table to the field that you just added. From this point, you can model the root table as you normally would.

Create a derived table

Next, create a derived table, and set the *SupportInheritance* property to *Yes*. Set the *Extends* property to point to the table on which the derived table is based. Set these properties before you create any fields for the table. This will help ensure that all fields in tables in the hierarchy have unique names and IDs, which is necessary for the run time to work correctly. It also makes it possible to choose different storage models, such as storing all types in a single table, without causing name collisions. Storage is discussed later in this section.

Work with existing tables

If tables already have fields before you add the tables to an inheritance hierarchy, you might need to update the field names and IDs in both metadata and code. If the tables already contain data, the existing data will need to be upgraded to work with the new table hierarchy. These are nontrivial tasks. For these reasons, creating a new table inheritance hierarchy from existing tables is not supported.

View the type hierarchy

You can use the Type Hierarchy Browser to view a table inheritance hierarchy. To do so, right-click a table in the AOT, and then click Add-Ins > Type Hierarchy Browser. Figure 17-11 shows the hierarchy for the FMVehicle table.

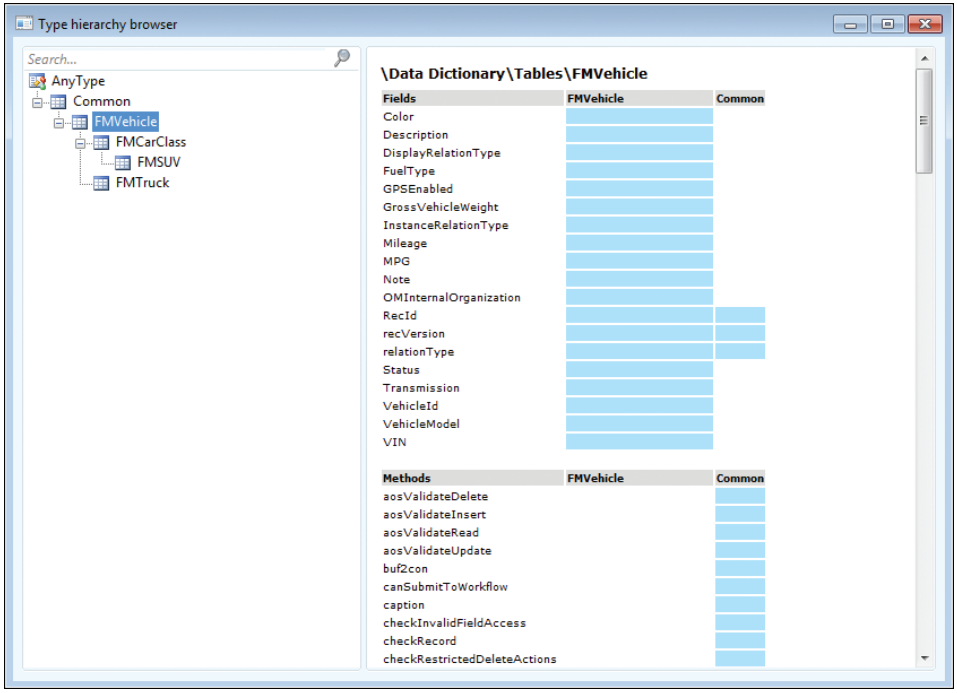


FIGURE 17-11 Hierarchy for the FMVehicle table.

Specify table behavior

Tables in an inheritance hierarchy share some property settings, so that table behavior is consistent throughout the hierarchy. These settings include the cache lookup mode, the OCC setting, and the save-data-per-company setting.

Configuration keys should be consistent with the table inheritance hierarchy. In other words, if a configuration key is disabled for the base table, the configuration key for the derived table should not be enabled. This condition is checked when you compile a table in the hierarchy, and errors are reported if the condition is found. For more information about configuration keys, see Chapter 11, "Security, licensing, and configuration."

You can specify whether a table in an inheritance hierarchy is concrete or abstract. By default, tables are concrete. This means that you can create a row that is of that table type. Specifying that a table is abstract means that you cannot create a row that is of that table type. Any row in the abstract table must be of a type of one of the derived tables (further up in the hierarchy) that is not marked as abstract. This concept aligns with the concept of an abstract class in an object-oriented programming language.

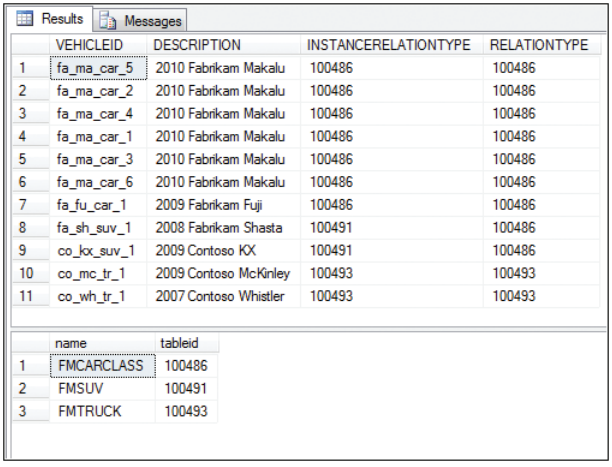
The table inheritance model in Microsoft Dynamics AX 2012 is a discrete model. Any row in the table hierarchy can be of only one concrete type (a table that is not marked as abstract). You cannot change the type of a row from one concrete type to another concrete type after the row is created.

Table inheritance storage model

In some implementations of object-relational (OR) mapping technologies, you can choose how the table inheritance hierarchy is mapped to data storage. The choices typically are one table for every modeled type, one table for every concrete type, or one table for every hierarchy. Microsoft Dynamics AX 2012 creates one table for every modeled type. Like a regular table, a table in a table inheritance hierarchy maps to a physical table in the database. Records in the inheritance hierarchy are linked through the *RecId* field. The data for a specific row of a type instance may be stored in multiple tables in the hierarchy, but they share the same *RecId*.

Every table in an inheritance hierarchy automatically has a system column that is named *RELATIONTYPE*. You will see this column in SQL Server, but not in the AOT. This column acts as a discriminator. The data for a concrete type is stored in multiple tables that make up the inheritance chain for that type. For a row in one of the tables, the discriminator column identifies the next table in the chain.

Figure 17-12 shows some rows in the FMVehicle table and the corresponding table IDs for its derived tables. The value of the *InstanceRelationType* field for cars equals the table ID of the FMCarClass table; for SUVs, the *InstanceRelationType* field equals the table ID of the FMSUV table; and for trucks, the *InstanceRelationType* field equals the table ID of the FMTrucks table. These represent the concrete type of each row in the FMVehicle table. The value of the *RelationType* field for both cars and SUVs equals the table ID of the FMCarClass table because FMCarClass is the next directly derived table for those rows. For trucks, the value of the *RelationType* field equals the table ID of the FMTruck table.


Results Messages

	VEHICLEID	DESCRIPTION	INSTANCERELATIONTYPE	RELATIONTYPE
1	fa_ma_car_5	2010 Fabrikam Makalu	100486	100486
2	fa_ma_car_2	2010 Fabrikam Makalu	100486	100486
3	fa_ma_car_4	2010 Fabrikam Makalu	100486	100486
4	fa_ma_car_1	2010 Fabrikam Makalu	100486	100486
5	fa_ma_car_3	2010 Fabrikam Makalu	100486	100486
6	fa_ma_car_6	2010 Fabrikam Makalu	100486	100486
7	fa_fu_car_1	2009 Fabrikam Fuji	100486	100486
8	fa_sh_suv_1	2008 Fabrikam Shasta	100491	100486
9	co_kx_suv_1	2009 Contoso KX	100491	100486
10	co_mc_tr_1	2009 Contoso McKinley	100493	100493
11	co_wh_tr_1	2007 Contoso Whistler	100493	100493

	name	tableid
1	FMCARCLASS	100486
2	FMSUV	100491
3	FMTRUCK	100493

FIGURE 17-12 Tables in the FMVehicle hierarchy.

Polymorphic behavior

When you issue a query on a table that is part of a table inheritance hierarchy, the Microsoft Dynamics AX run time provides the type fidelity by default. This means that if a *select* * statement is performed for a table, all of the rows of that table type are returned. For example, if you issue

the query *select * from DirPartyTable*, all instances of *DirParty* are returned, including *DirPerson*, *OperatingUnit*, and so on. Morever, because *select* * is used, the query returns complete data. This means that the DirPartyTable table must be joined to all of its derived tables.

When a *select* * is performed for a table that is part of a table inheritance hierarchy, that table is joined with an inner join to all of its base tables (all the way up to the root table), and then joined with an outer join to all of its derived tables (including derived tables at all levels). The reason for this is that any row in that table must have a corresponding row in all of its base tables, but could have matching rows in any of the concrete type paths. This ensures that, no matter what concrete type a row is, complete data for that row is always retrieved. Similar to the polymorphic behavior in object-oriented programming, this mechanism provides polymorphic data retrieval for a table that is part of a table inheritance hierarchy. In cases where you need to have all of the data for a row, this behavior is very convenient. You can use the dynamic method binding feature of table inheritance to write code that is clean and extensible.

For example, in the Fleet Management project, the FMVehicle table has a *doAnnualMaintenance* method that simply throws an exception. This happens because FMVehicle is an abstract table and any concrete table that is derived from it must override this method. The tables FMCarClass, FMTruck, and FMSUV all override this method, as shown in Figure 17-13. Note that each overridden method accesses a field that is not accessible from the base table.

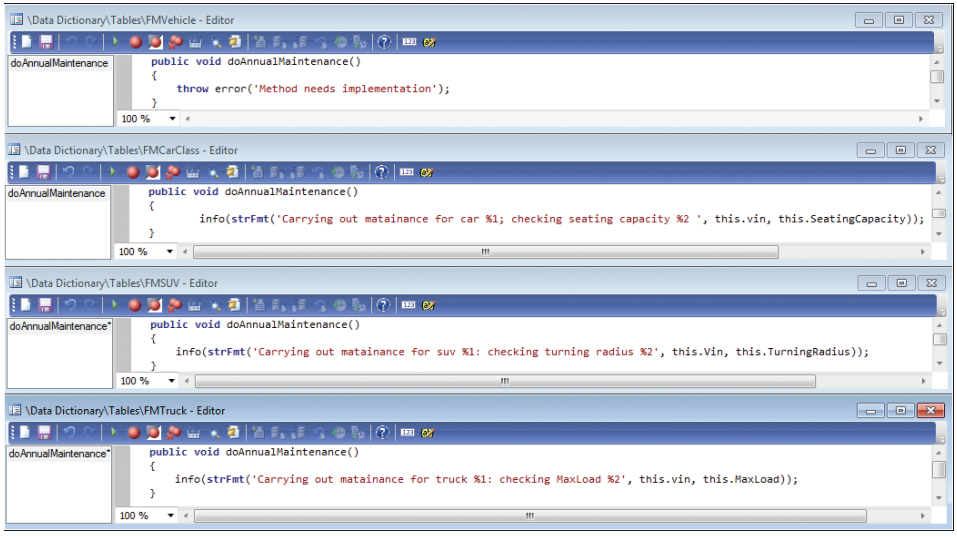


FIGURE 17-13 Overridden *doAnnualMaintenance* method on derived tables.

The following code queries the FMVehicle table and calls the *doAnnualMaintenance* method:

```
static void PolyMorphicQuery(Args _args)
{
    FMVehicle vehicle;

    while select vehicle
    {
```

```
        vehicle.doAnnualMaintenance();
    }
}
```

If you run the code as a job, you would get results that look similar to those shown in Figure 17-14.

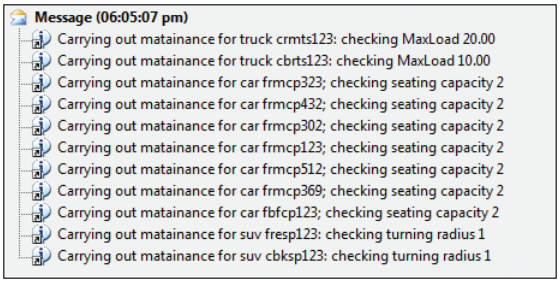


FIGURE 17-14 Result of calls to the overridden *doAnnualMaintenance* method.

As you can see, even though the *select* statement executes on the FMVehicle table, the statement returns fields in the derived tables. The actual *select* statement for this query looks like the following:

```
SELECT <all fields from all tables in the hiearchy> FROM  FMVEHICLE T1 LEFT OUTER JOIN FMTRUCK
T2 ON (T1.RECID=T2.RECID) LEFT OUTER JOIN FMCARCLASS T3 ON (T1.RECID=T3.RECID) LEFT OUTER JOIN
FMSUV T4 ON (T3.RECID=T4.RECID)
```

Tip You can get the Transact-SQL *select* statement directly from X++ code without having to use SQL Profiler. The following is an example:

```
static void PolyMorphicQuery(Args _args)
{
    FMVehicle vehicle;

    select generateonly vehicle;

    info(vehicle.getSQLStatement());
}
```

Performance considerations

When the inheritance hierarchy is very wide, very deep, or both, a polymorphic query can result in numerous table joins, which can degrade query performance. For example, the query *select * from DirPartyTable* produces eight table joins.

Exercise caution when using the table inheritance feature. Determine whether you really need all of the data from every derived type instance. If the answer is no, you should list the fields that you need explicitly in the field list. (Note that you can also list fields from derived tables when you model a query in the AOT or use the query object in X++ code. But you can only list fields from current and base tables when you write the X++ *select* statement.) The Microsoft Dynamics AX run time then adds joins to only the tables that contain the fields in the list. For example, if you change the query on the

DirPartyTable table to *select name from DirpartyTable*, only the DirPartyTable table is included in the query. No joins to other tables in the hierarchy are created because no data is being accessed from them. Careful query construction can improve query performance significantly.

Listing only the fields that are needed is not only beneficial here, but it is a good practice in general. This may allow SQL Server to use a covering index when processing the query and reduce the network load. A common concern about this practice is passing the table buffer to another function in another module, because you need to ensure that the other function does not use fields that were not selected to be returned. When an attempt is made to read fields that are not in the field list, Microsoft Dynamics AX 2009 and earlier versions does not produce an exception. You get whatever value is on the table buffer, which in most cases is an empty value. In Microsoft Dynamics AX 2012, an attempt to access an unavailable field raises an exception, but only if the field is included in a table that is part of a table inheritance hierarchy. A configuration setting is available to raise either a warning or exception for all tables that encounter this issue. To change this setting, do the following:

1. Click System Administration > Setup > System > Server Configuration.
2. On the Performance Optimization FastTab, under Performance Settings, click the drop-down list next to Error On Invalid Field Access to change the setting.

To maintain backward compatibility and to reduce run-time overhead, this setting is turned off by default. It is recommended that you activate this setting for testing purposes only.

Unit of Work

Maintaining referential integrity is important for any ERP application. Microsoft Dynamics AX 2012 lets you model table relations with richer metadata and express referential integrity in your data model more precisely. However, the application is still responsible for making sure that referential integrity is maintained. Microsoft Dynamics AX table relations are not implemented as database foreign key constraints in the SQL Server database. Implementing these constraints would add performance overhead in SQL Server. Also, for performance and other reasons, application code might violate referential integrity rules temporarily and fix the violations later.

Maintaining referential integrity requires performing data operations in the correct order. This is most prominent in cases where records are created and deleted. The parent record must be created first, before the child record can be inserted with the correct foreign key. But child records must be deleted before the parent record. Ensuring this manually in code can be error-prone, especially with the more normalized data model in Microsoft Dynamics AX 2012. Also, data operations are often spread among different code paths. This leads to extending locking and transaction spans in the database.

Microsoft Dynamics AX 2012 provides a programming concept called Unit of Work to help with these issues. Unit of Work is essentially a collection of data operations that is performed on related data. Application code establishes relationships between data in memory, modifies the data, registers the operation request with the Unit of Work framework, and then requests that the Unit of Work

perform all of the registered data operations in the database as a unit. Based on the relationships established among the entities in the in-memory data, the Unit of Work framework determines the correct sequence for the requested operations and propagates the foreign keys, if necessary.

The following code example shows Unit of Work in use:

```
public static void fmRentalAndRentalChargeInsert()
{
    FMTruck truck;
    FMRental rental;
    FMRentalCharge rentalCharge;
    FMCustomer customer;
    UnitofWork uoW;

    // Populate rental and RentalChange in UoW. 3 types of Rental Charge Records
    // for the same Rental.

    // Getting the customer and the truck that the customer is renting
    // These records are referred to from the rental record

    select truck where truck.VehicleId == 'co_wh_tr_1';
    select customer where customer.DriverLicense == 'S468-3184-6541';

    uoW = new UnitofWork();
    rental.RentalId = 'Redmond_546284';

    // This links the rental record with the truck record.
    // During insert, rental record will have the correct foreign key into the truck record.

    rental.fmVehicle(truck);

    // This links the rental record with the customer record.
    // During insert, rental record will have the correct foreign key into the
    // customer record.

    rental.fmCustomer(customer);
    rental.StartDate = DateTimeUtil::newDateTime(1\1\2008,0);
    rental.EndDate = DateTimeUtil::newDateTime(10\1\2008,0);
    rental.StartFuelLevel = 'Full';

    // Register the rental record with unit of work for save.
    // Unit of work makes a copy of the record.

    uoW.insertonSaveChanges(rental);

    uoW.saveChanges();

}
```

It is important to understand that Unit of Work copies the data changes into its own buffer when the registration method executes. After that, the original buffer is disconnected from Unit of Work. Any changes made to the table buffer after that will not be picked up by Unit of Work. If you want to save these changes through Unit of Work, you need to call the corresponding registration method again.

When you register multiple changes on the same record with Unit of Work, the last changes that are registered overwrite all previous changes.

In the previous code example, the code runs on the server because Unit of Work is a server-bound construct and cannot be instantiated or used on the client.

The form run time in Microsoft Dynamics AX uses the Unit of Work framework in its internal implementation to handle data operations on form data sources, where several form data sources are joined together directly. These scenarios did not work in previous releases of Microsoft Dynamics AX. When the form run time uses the Unit of Work framework, it is not accessible through X++.

Date-effective framework

Many business scenarios require tracking how data changes over a period of time. Some of the requirements include viewing the value of a record as it was in the past, viewing the value at the current time, or being able to enter a record that will become effective on a future date. A typical example of this is employee data in an application that is used by the Human Resources team of a company. Some questions that such an application will help answer might be as follows:

- What position did a specific employee hold on a specific date?
- What is the current salary for the employee?
- What is the current contact information for the employee?

In addition to answering these questions, there might also be a requirement to allow users to enter new data and change existing data. For example, a user could change the contact information for Employee C and make the new information effective on September 15, 2012. Such data is often referred to as date-effective data. Microsoft Dynamics AX 2012 supports creating and managing date-effective data in the database. The date-effective framework provides a number of features that include support for modeling entities, programmatic access for querying and updating date-effective data, run time support for maintaining data consistency, and user interface support. Core Microsoft Dynamics AX features, such as the Global Address Book and modules like Human Resources, use date-effective tables extensively in their data models.

Relational modeling of date-effective entities

Microsoft Dynamics AX provides support for modeling date-effective entities at the table level. The *ValidTimeStateFieldType* property of a table indicates whether the table is date-effective. This information is stored in the metadata for the table and is used at run time.

Figure 17-15 shows the DirPersonName table, which is used to track the history of a person's name. The table is date-effective because the *ValidTimeStateFieldType* property is set to *UtcDateTime*. When

you set this property on a table, the date-effective framework automatically adds the columns *ValidFrom* and *ValidTo*, which are required for every date-effective table. The data type of these columns is based on the value chosen for the *ValidTimeStateFieldType* property. Two data types are available:

- ***Date*** Tracking takes place at the day level. Records are effective starting from the *ValidFrom* date through the *ValidTo* date.
- ***UtcDateTime*** Tracking takes place at the second level. In this case, multiple records can be valid within the same day.

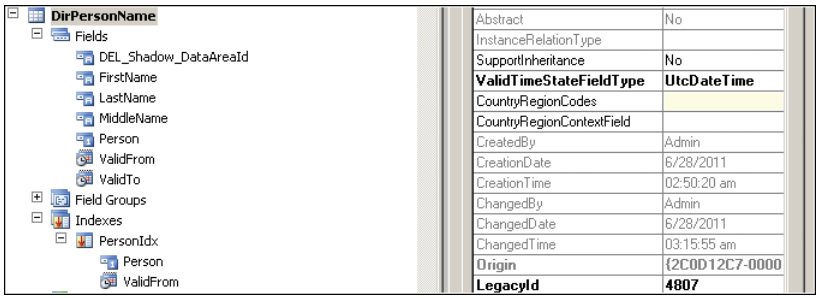


FIGURE 17-15 DirPersonName table with *ValidTimeStateFieldType* property set to *UtcDateTime*.

In addition to the fields each date-effective table is required to have, the table must have at least one alternate key that is implemented as a unique index. This alternate key is referred to as the *validtimestate* key in the date-effective framework, and it is used to enforce the time period semantics that are enabled by a date-effective table. The *validtimestate* key must contain the *ValidFrom* column and at least one other column other than the *ValidTo* column. The *validtimestate* key has an additional property that indicates whether gaps (missing records for a period of time) are allowed in the data. In Figure 17-16, the DirPersonName table is used track changes to the *Person* column. The *validtimestate* key contains the *Person* column and the *ValidFrom* column. When the *ValidTimeStateKey* property is set to *Yes* for an index, the index also needs to be a unique index and is required to be an alternate key.

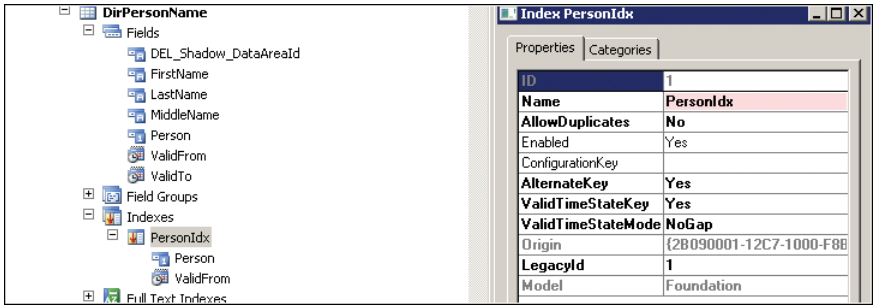


FIGURE 17-16 The *validtimestate* key index for the DirPersonName table.

In Table 17-2, the records reflect the changing names over a period of time for two people. The table must have a unique index with the following columns: *Person* and *ValidFrom*. The *ValidTo*

column can be part of the index, but it is optional. This index has the *ValidTimeStateKey* property set to *Yes*. Because the objective is to track the history of a person, the column that represents the person is also part of the *validtimestate* key. The *validtimestate* key enables the date-effective framework to indicate the field for which the history is being tracked.

TABLE 17-2 Tracking name changes over time in a date-effective table.

Person	FirstName	MiddleName	LastName	ValidFrom	ValidTo
1	Jim	M	Corbin	02/10/1983 00:00:00	04/16/1984 23:59:59
1	Jim	M	Daly	04/17/1984 00:00:00	12/31/2154 23:59:59
2	Anne		Wallace	04/14/2001 00:00:00	07/04/2005 23:59:59
2	Anne		Weiler	07/05/2005 00:00:00	12/31/2154 23:59:59

This scenario has a requirement not to allow gaps in each person's name data. To implement this requirement, the *ValidTimeStateMode* property is set to *NoGap*. For other scenarios, gaps in the data might be acceptable. This is also implemented by using the *ValidTimeStateMode* property. The date-effective framework also enforces that a person cannot have more than one name at the same time. This is called prevention of overlaps in the data. If the *ValidTo* column for a record contains the maximum value allowed (*12/31/215423:59:59*), it indicates that the record does not expire.

Support for data retrieval

Business application logic that is written in X++ may need to retrieve data that is stored in date-effective tables. To support this, the framework has three modes of data retrieval:

- **Current** Returns the record that is currently active by default, when you use a *select* statement or an application programming interface (API) to retrieve data from the table.
- **AsOfDate** Retrieves the record that is valid for the passed-in date or the *UtcDateTime* parameter. This can be in the past, current, or future. The *ValidFrom* column of the retrieved record is less than or equal to the value passed in. The *ValidTo* column is greater than or equal to the value passed in.
- **Range** Returns the records that are valid for the passed in range.

The X++ language supports a syntax that is similar to the Transact-SQL syntax that is used when querying relational databases. The date-effective framework enhances this syntax by adding the *validtimestate* keyword to indicate the type of query. The modes described earlier translate to the following queries.

Note The *ValidFrom* and *ValidTo* columns in these examples use the *Date* data type. If they used the *UtcDateTime* data type, the dates passed in would have to be in *UtcDateTime* format.

This query retrieves the current emergency contact information for *Employee A*. There is no need to specify any values for the *ValidFrom* and *ValidTo* columns in the *where* clause because the Microsoft Dynamics AX run time automatically adds them:

```
select * from EmployeeEmergencyContact where EmployeeEmergencyContact.Employee == 'A';
```

This query retrieves the record that was in effect on April 21, 1986:

```
select validtimestate (21\04\1986) * from EmployeeEmergencyContact where
EmployeeEmergencyContact.Employee == 'A';
```

This query retrieves all of the records for *Employee A* for the time period that is passed into the statement:

```
select validtimestate(01\01\1985, 31\12\2154)
* from EmployeeEmergencyContact where EmployeeEmergencyContact.Employee == 'A';
```

X++ also exposes a Query API to retrieve data from tables. This API has been extended with the following methods to allow different forms of querying:

- *validTimeStateAsOfDate(date)*;
- *validTimeStateAsOfDateTime(utcdatetime)*;
- *validTimeStateDateRange(date)*;
- *validTimeStateDateTimeRange(utcdatetime)*;

The date-effective framework uses these methods and transforms them by adding additional predicates on the *ValidFrom* and *ValidTo* columns to fetch the data that meets the requirement of the query.

Run-time support for data consistency

The data that is stored in a date-effective table must conform to the following consistency requirements:

- The data must not contain overlaps.
- Gaps are either allowed or disallowed, depending on the value of the *ValidTimeStateMode* property of the *validtimestate* key.

Because the data that is stored in the table can be added to or changed, the date-effective framework ensures that these consistency requirements are enforced. The date-effective framework implements these requirements by adjusting other records using the following rules:

- If *ValidFrom* is being updated, retrieve the previous record, and then update the *ValidTo* of the previous record to a value of *ValidFrom -1* to ensure that there is no overlap or gap. If the *ValidFrom* of the edited record is less than the *ValidFrom* of the previous record, display an error. The error is displayed because further action is required to delete the previous record

to avoid overlaps. The date-effective framework does not delete automatically records during adjustments.

- If *ValidTo* is being updated, retrieve the next record, and then update the *ValidFrom* of the next record to a value of *ValidTo + 1* to ensure that there is no overlap or gap. If the *ValidTo* of the edited record is greater than the *ValidTo* of the next record, display an error.
- The date-effective framework does not allow simultaneous editing of *ValidTo* and *ValidFrom* columns.
- The date-effective framework does not allow editing of any other columns that are part of the *validtimestate* key.
- When a new record is inserted, the *ValidTo* of the existing record is updated to a value of *ValidFrom -1* of the newly inserted record. New records cannot be inserted in the past or future if records already exist for that time period.
- When a record is deleted, the *ValidTo* of the previous record is adjusted to a value of *ValidFrom -1* of the next record, but only if the system requires that there should be no gaps. If gaps are allowed, no adjustment is performed.

Modes for updating records

The date-effective framework allows regular updates of records in a date-effective table. It also provides additional modes that are typical for the types of changes that are made to date-effective tables. The date-effective framework provides the following three modes:

- ***Correction*** This mode is analogous to regular updates to data in a table. If the *ValidFrom* or *ValidTo* columns are updated, the system updates additional records if necessary to guarantee that the data does not contain gaps or overlaps.
- ***CreateNewTimePeriod*** The date-effective framework creates a new record with updated values, and updates the *ValidTo* of the edited record to a value of *ValidFrom -1* of the newly inserted record. By default, the *ValidFrom* column of the newly inserted record has the current date for the *Date* data type columns or the current time for *UtcDateTime* data type columns. This mode does not allow editing of records in the past. This mode hides the date-effective characteristics of the data from the user. The user edits the records as usual, but internally a new record is created to continue tracking the history of changes made to the record.
- ***EffectiveBased*** This mode is a hybrid of the other two modes. Records in the past are prevented from being edited. Current active records are edited by using the same semantics as *CreateNewTimePeriod* mode. Records in the future are updated by using the same semantics as *Correction* mode.

The update mode must be specified by calling the *validTimeStateUpdateMode(ValidTimeState Update _validTimeStateUpdateMode)* method on the table buffer that is being updated. This method

takes a value from the *ValidTimeStateUpdate* enumeration as a parameter. This enumeration has the list of the various update modes.

Important The Microsoft Dynamics AX run time displays an error if the update mode is not specified when a date-effective table is updated through X++.

User experience

When the user updates a record in a date-effective table, other records might be updated as a consequence. The date-effective framework first provides a dialog box that informs the user about the additional updates and asks the user to confirm whether the action should proceed. The user's actions are simulated without actually updating the data. After the user chooses to update the data, the user interface is refreshed by retrieving all of the updated records.

Full-text support

Microsoft Dynamics AX 2012 has the capability to execute full-text search queries against the database. Full-text search functionality is provided by SQL Server and allows the ability to perform linguistic searches against text data stored in the database. For more information, see "Full-Text Search (SQL Server)" at *http://msdn.microsoft.com/en-us/library/ms142571.aspx*.

Microsoft Dynamics AX provides the capability to create a full-text index on a table. New methods that are available in the *Query* class that let you write queries that use this index. Figure 17-17 shows a full-text index that has been created for the VendTable table in the AOT.

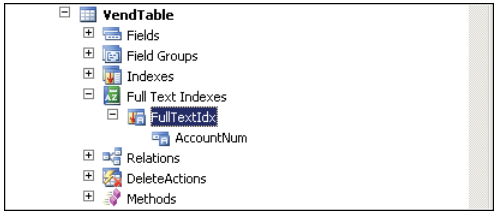


FIGURE 17-17 Full-text index for the VendTable table.

Only one full-text index can be created for a table. Only fields that have the *string* data type can be used as fields for the full-text index. When the table is synchronized, a corresponding full-text index is created in the database. Microsoft Dynamics AX also requires that the table be part of either the Main table group or the Group table group. You cannot create a full-text index for temporary tables.

The following example shows how full-text queries can be executed by using the Query API:

```
Query q;
QueryBuildDataSource qbds;
```

```
QueryBuildRange qbr;
QueryRun qr;
VendTable vendTable;

q = new Query();
qbds = q.addDataSource(tablenum(VendTable));
qbr = qbds.addRange(fieldnum(VendTable, AccountNum));
qbr.rangeType(QueryRangeType::FullText);
qbr.value(queryvalue('SQL'));
qr = new QueryRun(q);

while (qr.next())
{
   vendTable = qr.get(tablenum(VendTable));
   print vendTable.AccountNum;
}
```

The *QueryRangeType::FullText* enumeration used by the *rangeType* method causes the data layer to generate a full-text search query in the database. Microsoft Dynamics AX uses the *FREETEXT* keyword provided by SQL Server when it generates a full-text query that is executed on the database. For the previous code example, the following Transact-SQL query is generated:

```
SELECT T1.ACCOUNTNUM,T1.INVOICEACCOUNT,…
FROM VENDTABLE T1 WHERE (((PARTITION=?) AND (DATAAREAID=?)) AND (FREETEXT(ACCOUNTNUM,?))) ORDER
BY T1.ACCOUNTNUM
```

For more information about the *FREETEXT* keyword, see "Querying SQL Server Using Full-Text Search" on MSDN at *http://msdn.microsoft.com/en-us/library/ms142559.aspx*.

You can also use the extended query range syntax to generate a full-text query. This is shown in the following example. The *freetext* keyword specifies that the data layer should generate a full-text query.

```
qbrCustDesc = qsbdCustGrp.addRange(fieldnum(VendTable, AccountNum));
qbrCustDesc.value('((AccountNum freetext "bike") || (AccountNum freetext "run"))');
```

The QueryFilter API

A favorite Transact-SQL interview question is to ask a candidate to explain the difference between the *on* clause and the *where* clause in a *select* statement that involves joins. You can find the long version of the answer on MSDN, which talks about the different logical phases of query evaluation. The short answer is that there is no difference between the two for an inner join. For an outer join, rows that do not satisfy the *on* clause are included in the result set, but rows that do not satisfy the *where* clause are not.

So you might wonder when to use *on* and when to use *where*. An example will help illustrate. The following polymorphic query finds all DirParty instances with the name *John*. There are two kinds of

predicates here. The first one matches the individual rows with their corresponding base or derived type:

```
<baseTable>.recID == <derivedTable>.recid.
```

The second predicate matches the name:

```
DirPartyTable.name == 'John'
```

To start with the first predicate, when you take a specific row from the root table, it may have a matching row in any one of the derived tables. Because the goal is to retrieve complete data for all types, you do not want to discard a row just because it does not match one of the derived tables. For this reason, you want to use the *on* clause. For the second predicate, you want only the rows that qualify the predicate. Thus, you want to use the *where* clause. The Transact-SQL for the query looks like this:

```
SELECT * FROM  DIRPARTYTABLE T1 LEFT OUTER JOIN DIRPERSON T2 ON (T1.RECID=T2.RECID) LEFT OUTER
JOIN DIRORGANIZATIONBASE T3 ON (T1.RECID=T3.RECID) LEFT OUTER JOIN DIRORGANIZATION T4 ON (T3.
RECID=T4.RECID) LEFT OUTER JOIN OMINTERNALORGANIZATION T5 ON (T3.RECID=T5.RECID) LEFT OUTER JOIN
OMTEAM T6 ON (T5.RECID=T6.RECID) LEFT OUTER JOIN OMOPERATINGUNIT T7 ON (T5.RECID=T7.RECID) LEFT
OUTER JOIN COMPANYINFO T8 ON (T5.RECID=T8.RECID) WHERE (T1.NAME='john')
```

You might notice that the *on* clause is specified immediately after the table join, and the *where* clause is specified at the end of the query after all of the table joins and *on* clauses. This matches the order in which the query is evaluated. The *where* clause predicates are evaluated after all of the joins have been processed. The following X++ *select* statement produces a similar query:

```
static void Job3(Args _args)
{
    SalesTable so;
    SalesLine  sl;

    select so where so.CustAccount == '1101'
        outer join sl where sl.SalesId == so.SalesId;
}
```

The Transact-SQL for the query looks like this:

```
SELECT * FROM  SALESTABLE T1 LEFT OUTER JOIN SALESLINE T2 ON ((T2.DATAAREAID=N'ceu') AND (T1.
SALESID=T2.SALESID)) WHERE ((T1.DATAAREAID=N'ceu') AND (T1.CUSTACCOUNT=N'1101'))
```

There is something of a mix here. The keyword is *where*, but it is specified after each table join. So where does the predicate go in the Transact-SQL? If you use SQL trace, you'll see that for an outer join, the predicates appear in the *on* clause. For an inner join, it shows in the *where* clause. To understand this, keep in mind that the X++ *where* is actually the *on* clause in Transact-SQL. Because there is no difference between *on* and *where* for inner joins, the Microsoft Dynamics AX run time simply moves those to the *where* clause. The X++ Query programming model has the same behavior. Query ranges that you specify by using the *QueryBuildRange* clause go in the *on* clause.

So how do you specify *where* clause predicates? For the X++ *select* statement, you may be able to attach these *where* clause predicates on one of the tables that are inner-joined. Alternatively, you could add these to the *where* clause of the first table if all the other tables are outer-joined. The solution is more difficult with the Query programming model because you cannot specify *QueryBuildRange* on another data source without using the extended query range feature. To solve this problem, Microsoft Dynamics AX 2012 added support for the *QueryFilter* API.

Because the *where* clause is evaluated at the query level after all of the joins have been evaluated, the *QueryFilter* API is available at the query level. You can refer to any query data source that is not part of an exists/notexists subquery. The following example shows the use of *QueryFilter*:

```
public void setFilterControls()
{
    Query query = fmvehicle_qr.query(); // Use QueryRun's Query so that the filter can be
cleared
    QueryFilter filter;
    QueryBuildRange range;
    boolean useQueryFilter = true;  // Change to false to see QueryRange on outer join

    if (useQueryFilter)
    {
        filter = this.findOrCreateQueryFilter(
            query,
            query.dataSourceTable(tableNum(FMVehicleMake)),
            fieldStr(FMVehicleMake, Make));
        makeFilter.text(filter.value());
    }
    else
    {
        // Optional code to illustrate behavior difference
        // between QueryFilter and QueryRange
        range = SysQuery::findOrCreateRange(
            query.dataSourceTable(tableNum(FMVehicleMake)),
            fieldNum(FMVehicleMake, Make));
        makeFilter.text(range.value());
    }
}

public QueryFilter findOrCreateQueryFilter(
    Query _query,
    QueryBuildDataSource _queryDataSource,
    str _fieldName)
{
    QueryFilter filter;
    int i;

    for(i = 1; i <= _query.queryFilterCount(); i++)
    {
        filter = _query.queryFilter(i);
        if (filter.dataSource().name() == _queryDataSource.name() &&
            filter.field() == _fieldName)
```

```
        {
            return filter;
        }
    }

    return _query.addQueryFilter(_queryDataSource, _fieldName);
}
```

QueryFilter has similar grouping rules about how individual predicates are constructed with *AND* or *OR* operators. First, *QueryFilters* are grouped by query data source and the results are combined by using the *AND* operator. Next, within a group for a specific query data source, the same rules for *QueryRange* are applied: predicates on the same fields use the *OR* operator first and then the *AND* operator.

If you run the following X++ code and look at the Transact-SQL trace, you will see CROSS JOIN in the Transact-SQL statement. You may think that it misses the join condition and is doing a Cartesian product of the two tables. The join condition is actually in the *where* clause. A cross join like this is equivalent to an inner join with the join condition. The cross join is needed because the two tables must be listed before the outer join tables, because they appear as outer join conditions. Transact-SQL does not allow you to reference tables before they are used in the query.

```
static void CrossJoin(Args _args)
{
    CustTable cust;
    SalesTable so;
    SalesLine sl;

    select generateonly cust where cust.AccountNum == '*10*'
         join so where cust.AccountNum == so.CustAccount
            outer join sl where so.SalesId == sl.SalesId;

    info(cust.getSQLStatement());
}
```

The Transact-SQL for the query looks like this:

```
SELECT * FROM CUSTTABLE T1 CROSS JOIN SALESTABLE T2 LEFT OUTER JOIN SALESLINE T3 ON (((T3.
PARTITION=?) AND (T3.DATAAREAID=?)) AND (T2.SALESID=T3.SALESID)) WHERE (((T1.PARTITION=?) AND
(T1.DATAAREAID=?)) AND (T1.ACCOUNTNUM=?)) AND (((T2.PARTITION=?) AND (T2.DATAAREAID=?)) AND (T1.
ACCOUNTNUM=T2.CUSTACCOUNT))
```

Data partitions

In previous releases of Microsoft Dynamics AX, the *DataAreaId* column in a table was used to provide the data security boundary. It was also used to define legal entities through the concept of a company. As part of the organizational model and data normalization changes, a large number of entities like Products and Parties that were previously stored per-company, have been updated to be shared (through global tables) in Microsoft Dynamics AX 2012. This was done primarily to enable sharing of data across legal entities and to avoid data inconsistencies.

But in some deployments of Microsoft Dynamics AX, data is not expected to be shared across legal entities. In such deployments, the *DataAreaId* column is primarily used as a security boundary to segregate data into various companies. Such customers want to share the deployment, implementation, and maintenance cost of Microsoft Dynamics AX, but they have no other shared data or shared business processes. There are also holding companies that grow by acquiring independent businesses (subsidiaries), but the data and processes are not shared among these subsidiaries.

Microsoft Dynamics AX 2012 R2 provides a solution to these requirements. This release of Microsoft Dynamics AX uses the concept of data partitioning by adding a *Partition* column to the tables in the database. This allows the data to be truly segregated into separate partitions. When a user logs into Microsoft Dynamics AX, he or she always operates in the context of a specific partition. The system ensures that data from only the specified partition is visible, and that all business processes run in the context of that specific partition.

Partition management

The Partitions table contains the list of partitions that are defined for the system. During setup, Microsoft Dynamics AX creates a default partition called *initial*. A system administrator can create additional partitions by using the Partitions form in the System Administration module.

Development experience

A new property named *SaveDataPerPartition* has been added for all tables in the AOT. By default, the value is set to *Yes*, and the property cannot be edited. This property can be edited only if the *SaveDataPerCompany* property is set to *No* and the table is marked as a SystemTable, or if the table belongs to the Framework table group. These checks are put in place to enable all application tables to be partitioned. Only specific tables that are used by the kernel can have data that is not partitioned.

All of the tables whose *SaveDataPerPartition* property is set to *Yes* have a Partition system column in the metadata. In the database, the table has a *PARTITION* column with a data type *int64*. It is a surrogate foreign key to the RECID column of the PARTITION table. This column always contains a value from one of the rows in the PARTITION table. The column has a default constraint with the RECID value of the initial partition. The kernel adds the *PARTITION* column to all the indexes in a partition-enabled table except for the RECID index.

Run-time experience

The Microsoft Dynamics AX kernel framework handles the population and retrieval of the *Partition* column based on the partition that is specified for the current session. The various database operations provided by Microsoft Dynamics AX have the following functionality:

- ***select* statements** All *select* statements are filtered automatically based on the current partition of the session. The Transact-SQL statement that is generated always has the *Partition* column in the *WHERE* clause.
- ***insert* statements** The inserted buffer always has the *Partition* column set to the partition of the current session. Microsoft Dynamics AX displays an error if the application code sets the column to a value that is different from the current partition of the session.
- ***update* statements** All updates are performed in the current partition. Updating the *Partition* column is not allowed.

Because of this functionality, you usually do not have to write code to handle the *Partition* column. However, an exception is any code that uses direct Transact-SQL. The *Partition* column will not be handled automatically, and the direct SQL code has to ensure that the *WHERE* clause contains the partition of the current session.

To provide strict data isolation, the framework does not provide the ability to change partitions programmatically at run time. To change the partition, a new session has to be created that is set to use the other partition. Certain framework components like Setup and Batch use the *runAs* method, which creates a new session to execute code in a different partition. This is not a common pattern and should not be used in non-framework application logic.

Similarly, the framework does not allow execution of database operations that span multiple partitions. This is a contrast from the cross-company functionality that allows execution of database statements across multiple legal entities.

CHAPTER 18

The batch framework

In this chapter

Introduction 613
Batch processing in Microsoft Dynamics AX 2012 613
Create and execute a batch job 615
Manage batch execution 625
Debug a batch task 629

Introduction

The batch framework in Microsoft Dynamics AX 2012 is an asynchronous, server-based task execution environment. It lets users execute asynchronous tasks in parallel and across multiple instances of the Application Object Server (AOS). In this release, the batch framework has been further enhanced from Microsoft Dynamics AX 2009. Among other enhancements, the batch framework now runs code in .NET common intermediate language (CIL), gives system administrators and developers increased control over batch jobs, and enables better performance and greater reliability when those jobs execute.

Microsoft Dynamics AX 2012 includes several tools that support the batch. The Batch Job form gives system administrators increased flexibility in the design, setup, and execution of complex batch jobs. In addition, the Batch Job form provides the ability to add multiple batch tasks to a single batch job and to define the dependencies among those tasks. An enhanced Batch application programming interface (API) gives X++ developers more control over complex batch jobs, along with the ability to process batch jobs directly from business logic.

A new framework, the SysOperation framework, was also introduced in this release. This framework enables application logic to be written in a way that supports running interactively or via the batch framework. It is a refinement of the RunBase framework and provides additional flexibility for creating new batch jobs. For more information about the SysOperation framework, see Chapter 14, "Extending Microsoft Dynamics AX."

Batch processing in Microsoft Dynamics AX 2012

Batch processing is a noninteractive task-processing technique where users create batch jobs to organize appropriate types of tasks to be processed as a unit. Batch processing has some important advantages: it lets users schedule batch tasks and define the conditions under which they execute, add the tasks to a queue, and set them to run automatically on a batch server. After execution is

complete, the batch server logs any errors and sends alerts. A batch job might involve printing reports, closing inventory, or performing periodic maintenance. By scheduling a batch job to process these types of resource-intensive tasks in off-peak hours, users can avoid slowing down the system during working hours.

Table 18-1 describes how standard batch processing concepts are represented in Microsoft Dynamics AX. These concepts are discussed in greater detail throughout this chapter.

TABLE 18-1 Batch processing concepts in Microsoft Dynamics AX 2012.

Concept	Description
Batch task	The smallest unit of work that can be executed using the batch framework. It is a batch-executable class that contains business logic to perform a certain action. The Microsoft Dynamics AX classes that are used for batch tasks are designated to run on the server. These tasks can run automatically as part of a batch job on the AOS. This version of the product has limited support for client batch jobs; it is recommended that you use server-side batch jobs to take full advantage of the new features in Microsoft Dynamics AX 2012. For more information, see the "Create a batch-executable class" section, later in this chapter.
Batch job	A complete process that achieves a goal, such as printing a report or performing the inventory closing process. A batch job is made up of one or more batch tasks.
Batch group	A logical categorization for batch tasks that lets administrators specify which AOS instance runs a particular task. Tasks that are not explicitly assigned to a batch group are, by default, assigned to an *empty* (default) group.
Batch server	An AOS instance that processes batch jobs. Read more about AOS in Chapter 1, "Architectural overview." For more information about configuring an AOS instance to be a batch server, see the "Configure the batch server" section, later in this chapter.

Common uses of the batch framework

Organizations can use the batch framework to perform asynchronous operations in a variety of scenarios. Typically, organizations create batch jobs to address the following kinds of needs:

- **Enable scheduling flexibility** The batch framework can perform periodic tasks on a regular schedule, such as data cleanup or invoice processing. For example, to run invoice processing at the end of every month, you can set up a recurring batch job that runs at midnight on the last working day of each month. The batch framework automatically picks up the job and processes pending invoices according to the specified schedule.
- **Control the order in which tasks execute** With the batch framework, you can develop a workflow or perform a complex data upgrade in a sequence that you specify. You can also set up dependencies between the tasks and create a dependency tree that ensures that certain tasks run in sequence while others run in parallel.
- **Enable conditional processing** Decision trees can help you implement a reliable way of processing data. Developers or system administrators can set up dependencies between tasks in such a way that different tasks are executed, depending on whether a particular task succeeds or fails. (Figure 18-4, later in the chapter, shows an example of a dependency tree.) System administrators can also set up alerts so that they are notified if a job fails.

- **Improve performance by using parallelization** The batch framework lets you take advantage of multithreading, which ensures that your processor's capabilities are used fully. This is particularly important for long-running processes, such as inventory closing. You can improve performance further by breaking a process into tasks and executing them against different AOS instances, thus increasing the throughput and reducing overall execution time.
- **Implement advanced logging and profiling** The batch framework lets you see what errors or exceptions were thrown the last time the batch ran, and it also shows you how long a process takes to execute. Advanced logging and the new profiling capabilities are also useful for performance benchmarking and security auditing.

Performance

The new capability to run larger and more complex batch jobs has required performance enhancements to the batch framework. In Microsoft Dynamics AX 2012, the batch framework is designed to be a server-side component. This lets you design multithreaded server processes in a controlled manner. By configuring the number of parallel execution threads and servers, defining the set and order of tasks for processing, and setting the execution schedule, you can achieve greater scalability across your hardware.

As mentioned earlier, the batch framework is now designed to run X++ that has been compiled as .NET CIL code for batch jobs. This significantly improves performance compared to Microsoft Dynamics AX 2009, which ran interpreted X++ code. Compared to the interpreted code, garbage collection is much better, and because of session pooling, scheduling new batch jobs is less resource intensive. You can also profile the performance of jobs by using Microsoft Visual Studio Performance Profiler.

Microsoft developers use the batch framework as a foundation for many performance-critical processes, such as maximizing hardware scalability during a data upgrade and maximizing throughput during journal posting. For more information about how Microsoft uses the batch framework for performance-critical processes, see the white paper "Journal Batch Posting," available at *http://www.microsoft.com/en-us/download/details.aspx?id=13379*.

Create and execute a batch job

Microsoft Dynamics AX 2012 includes numerous batch jobs that perform operations such as generating reports, creating sales invoices, and processing journals. However, in several situations, organizations need to create their own batch jobs. The batch framework provides full flexibility in the types of jobs that you can create. This section walks you through the following steps, which are required for creating, executing, and managing a batch job:

1. Create a batch-executable class.
2. Create a batch job and define the execution schedule.

3. Configure a batch server and create a batch group.
4. Manage the batch job.

Create a batch-executable class

The first step in developing a batch job is to define a class that can be executed as a batch task. Many classes included with Microsoft Dynamics AX 2012 are already enabled for batch processing. You can also design a batch-executable class, as shown in the following example:

```
public class ExampleBatchTask extends RunBaseBatch
```

To run as a batch task, a class must implement the Batchable interface. The best way to implement the interface contract is to extend the *RunBaseBatch* abstract class, which provides much of the necessary infrastructure for creating a batch-executable class. An alternative is to use the SysOperation framework, which provides additional advantages compared to extending the *RunBaseBatch* class. For more information about the SysOperation framework, see Chapter 14.

The *RunBaseBatch* class is an extension of the RunBase framework, so your batch class must adhere to the patterns and guidelines of the *RunBase* extended classes (see Chapter 14 for more information).

Table 18-2 describes the methods that must be implemented when you extend the *RunBaseBatch* class. The following sections describe these methods in more detail.

TABLE 18-2 Required methods for extensions to the *RunBaseBatch* class.

Method	Description
run	Contains the core logic for your batch task
pack	Serializes the list of variables used in the class
unpack	Deserializes the list of variables used in the class
canGoBatchJournal	Determines whether the class appears in the Batch Task form

run method

You implement the core logic of your batch class in the *run* method. The *run* method is called by the batch framework for executing the task defined within it. You can run most of the X++ code in this method; however, there are some limitations on the operations that you can implement. For example, you can't call any client logic or dialog boxes. However, you can still use the *Infolog* class. All Infolog and exception messages are captured when the batch class executes, and they are stored in the batch table. You can view these later in the Batch Job form or the Batch Job History form, both of which are located under System Administration > Inquiries > Batch Jobs.

Note If an error message is written to the Infolog, it does not mean that the task has failed; instead, an exception must be thrown to indicate the failure.

pack and *unpack* methods

A class that extends *RunBaseBatch* must also implement the *pack* and *unpack* methods to enable the class to be serialized. When a batch task is created, its member variables are serialized by using the *pack* method and stored in the *batch* table. Later, when the batch server picks up the task for execution, it deserializes class member variables by using the *unpack* method. So it's important to provide a correct list of the variables that are necessary for class execution. If any member variable isn't packable, then the class can't be serialized and deserialized to the same state.

The following example shows the implementation of the *pack* and *unpack* methods:

```
public container pack()
{
    return [#CurrentVersion,#CurrentList];
}

public boolean unpack(container _packedClass)
{
    Version version     = RunBase::getVersion(_packedClass);
    switch (version)
    {
        case #CurrentVersion:
            [version,#CurrentList] = _packedClass;
            break;
        default:
            return false;
    }
    return true;
}
```

The *#CurrentList* and *#CurrentVersion* macros that are referenced in the preceding code must be defined in the class declaration. Using a macro simplifies the management of variables in the class. If you add or remove variables later, you can manage the list by modifying the macro. The *#CurrentList* macro holds a list of the class member variables to pack, as shown here:

```
    #define.CurrentVersion(1)
    #localmacro.CurrentList
            methodVariable1,
            methodVariable2
    #endmacro
```

canGoBatchJournal method

When a system administrator creates a new batch task by using the Batch Task form, the *canGoBatchJournal* method determines whether the batch task class appears in the list of available classes. For an example of how to use *canGoBatchJournal*.

Create a batch job

The second step in developing a batch job is to create the batch job and add batch tasks. You can create a batch job in three ways:

- By using the dialog box of a batch-enabled class
- By using the Batch Job Designer form
- By using the Batch API

The method you use depends on the degree of flexibility that you need and the complexity of the batch job. To create a simple batch job, consisting of a single task with no dependencies, you typically use the dialog box of a batch-executable class; to create a more complex batch job, consisting of several tasks that might have dependencies, use the Batch Job form; to create a highly complex or very large batch job, or one that needs to be integrated with other business logic, use the Batch API. The following sections provide an example of using each method.

Create a batch job from the dialog box of a batch-executable class

The simplest way to run a batch-executable class as a batch job is to invoke the class by using a menu item. A menu item that points to a batch-executable class automatically opens a dialog box that lets the user create a batch job. On the Batch tab of the dialog box, select the Batch Processing check box, as shown for the *Change based alerts* class in Figure 18-1. When you select Batch Processing and click OK, a new batch job with the task that represents the batch-executable class is created. The batch job then runs asynchronously at the date and time you specify. You can also set up recurrences or alerts for the job by clicking the appropriate button on the right side of the dialog box. You can also specify the batch group for the task by using the drop-down list.

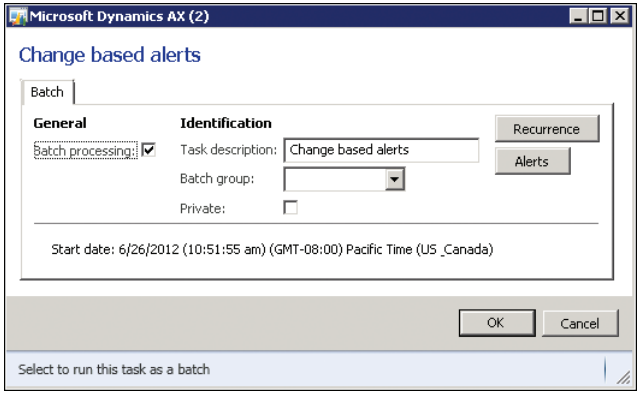


FIGURE 18-1 An example of the Batch tab for a class.

Create a batch job by using the Batch Job form

You can open the Batch Job form from several places. For example, you can open it by clicking Batch Jobs from System Administration > Inquiries > Batch Jobs or by selecting My Batch Jobs (for users) from Home > Inquiries > My Batch Jobs. Both menu items open the same form, but the information that is presented in the form differs, depending on the menu item that you use to open it. Depending on how you open the form and your level of access, you can view either the batch jobs that you have created or all batch jobs that are scheduled in the system.

Press Ctrl+N to create a new batch job, and then enter the details for the job in the grid or on the General tab: a description, and the date and time at which you want the job to start. You can also set up recurrence for the batch job by clicking Recurrence on the menu bar, and then entering a range and pattern for the recurrence.

Note If you don't enter a date and time, the current date and time are entered automatically.

Figure 18-2 shows the Batch Job form.

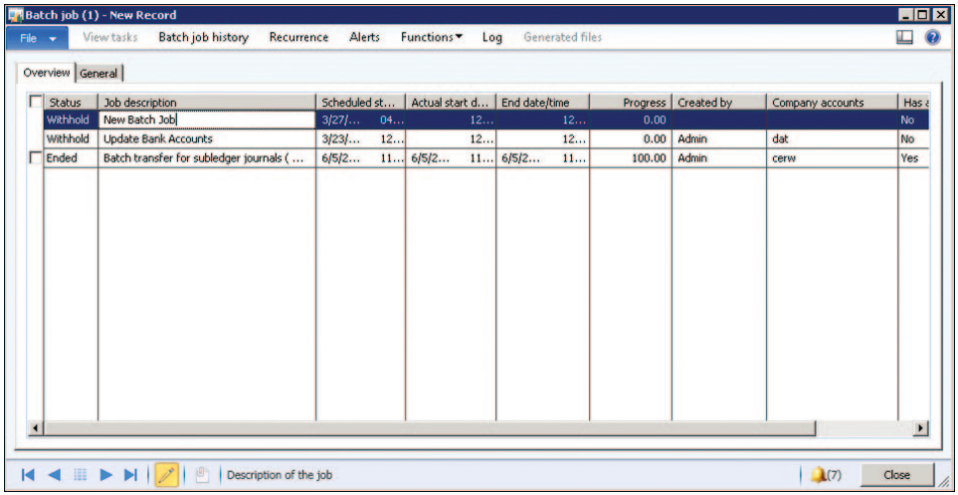


FIGURE 18-2 The Batch Job form.

After you create a batch job, you can add tasks to it and create dependencies between them by using the Batch Tasks form (shown in Figure 18-3). The Batch Tasks form opens when you click View Tasks on the menu bar in the Batch Job form. From the Batch Tasks form, you can also change the status of batch tasks or delete tasks that are no longer needed.

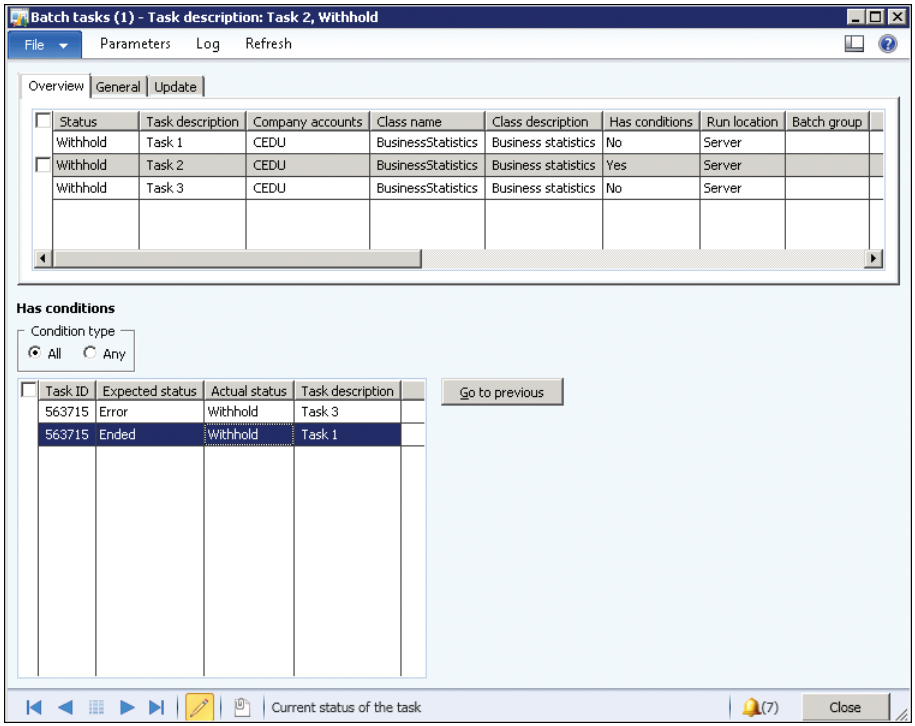


FIGURE 18-3 The Batch Tasks form.

To create a task, do the following:

1. Press Ctrl+N to create the task.
2. In Task Description, enter a description of the task.
3. In Company Accounts, select the company in which the task runs.
4. In Class Name, select the process that you want the task to run. Classes appear in a lookup list containing all available batch-enabled classes. The lookup list appears only if the *CanGoBatchJournal* property is enabled.
5. In Batch Group, select a batch group for the task if necessary.
6. Save the task by pressing Ctrl+S.
7. Specify class parameters if necessary. As mentioned in previous sections, each batch task represents a batch-executable class. Sometimes you need to set up parameters for that class. For example, you might need to specify posting parameters for invoice posting. To do that, click Parameters on the menu bar in the Batch Tasks form. A dialog box specific to the selected class is displayed.

Note If you are creating a custom batch class, you must design the parameters form manually. If you implement a batch based on the SysOperation framework, this process is highly simplified. After you specify the necessary parameters and click OK, the class parameters are packed and saved in the *Batch* table and then are restored when the class executes. For more information about the SysOperation framework, see Chapter 14.

8. Set up dependencies or advanced sequencing between tasks, if necessary.

After you create the batch job and add tasks to it, you can use the Batch Tasks form to define dependencies between the tasks. If no dependencies or conditions are defined within a job, the batch server automatically executes the tasks in parallel. (To configure the maximum number of parallel tasks, use the Maximum Batch Threads parameter in the Server Configuration form.)

If you need to use advanced sequencing to accommodate your business process flow, you can use either the Batch Tasks form or the Batch API. You can use these tools to construct complex dependency trees that let you schedule batch jobs tasks in parallel, add multiple dependencies between batch tasks, choose different execution paths based on the results of the previous batch task, and so on.

For example, suppose that the job, JOB1, has seven tasks: TASK1, TASK2, TASK3, TASK4, TASK5, TASK6, and TASK7, and you want to set up the following sequence and dependencies for it:

- TASK1 runs first.
- TASK2 runs on completion (Ended or Error) of TASK1 (regardless of the success or failure of TASK1).
- TASK3 runs on success (Ended) of TASK2.
- TASK4 runs on success (Ended) of TASK2.
- TASK5 runs on failure (Error) of TASK2.
- TASK6 runs on failure (Error) of TASK3.
- TASK7 runs on success (Ended) of both TASK3 and TASK4.

Figure 18-4 shows the dependency tree for JOB1.

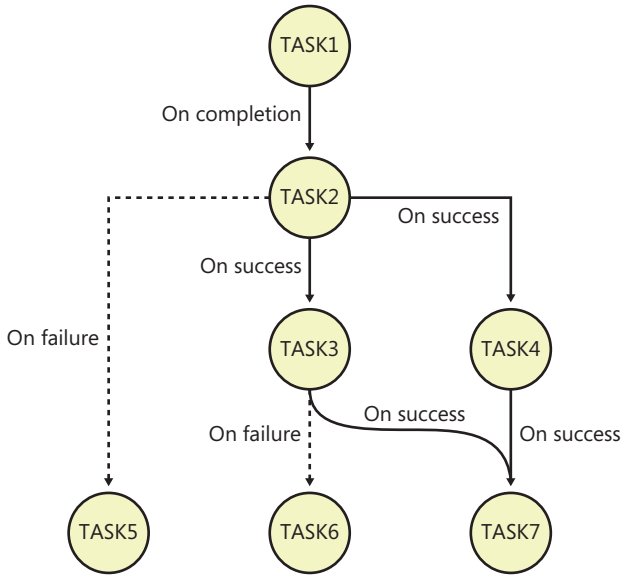


FIGURE 18-4 The batch task dependency tree for JOB1.

To define these task dependencies and to tell the system how to handle them, select a child task—for example, TASK2—from the preceding list, and then do the following:

1. In the Batch Tasks form, click in the Has Conditions grid, and then press Ctrl+N to create a new condition.
2. Select the task ID of the parent task, such as TASK1.
3. Select the status that the parent task must reach before the dependent task can run. For example, TASK2 starts when the status of TASK1 becomes Ended or Error.
4. Press Ctrl+S to save the condition.
5. If you enter more than one condition, and if all conditions must be met before the dependent task can run, select a condition type of All.

   Alternatively, if the dependent task can run after any of the conditions are met, select a condition type of Any.

You can use the Batch Tasks form to define how the system handles task failures. To ignore the failure of a specific task, select Ignore Task Failure for that task on the General tab. If you select this option, the failure of the task doesn't cause the job to fail. You can also use Maximum Retries to specify the number of times a task should be retried before it fails.

Use the Batch API

For advanced scenarios requiring complex or large batch jobs, such as inventory closing or data upgrades, the batch framework provides an X++ API that you can use to create or modify batch jobs, tasks, and their dependencies as needed; and to create runtime batch tasks dynamically. This flexible API helps you automate task creation and integrate batch processing into other business processes. It can also be useful when your batch job or task requires additional logic. To create a batch job by using the Batch API, the following steps are necessary:

1. Use the *BatchHeader* class to create the batch job.
2. Modify the parameters for the batch job.
3. Add tasks.
4. Define the dependencies between tasks.
5. Save the batch job.

Create a batch job by using the *BatchHeader* class

Create an instance of the *BatchHeader* class that represents your batch job. The following example creates a *BatchHeader* instance named *sampleBatchHeader*:

```
sampleBatchHeader = BatchHeader::construct();
```

You can also construct a *BatchHeader* object for an existing batch job by providing an optional *batchJobId* parameter for the *construct* method, as shown here:

```
//job1 is an existing job
sampleBatchHeader = BatchHeader::construct(job1.parmCurrentBatch().BatchJobId);
```

Modify batch job parameters

The *BatchHeader* class lets you access and modify most parameters by using *parm* methods. For example, you can set up recurrences and alerts for your batch job, as shown in the following example:

```
// Set the batch recurrence
sysRecurrenceData = SysRecurrence::defaultRecurrence();
sysRecurrenceData = SysRecurrence::setRecurrenceStartDateTime(sysRecurrenceData,
DateTimeUtil::utcNow());
sysRecurrenceData = SysRecurrence::setRecurrenceNoEnd(sysRecurrenceData);
sysRecurrenceData = SysRecurrence::setRecurrenceUnit(sysRecurrenceData, SysRecurrenceUnit::Hour,
1);
sampleBatchHeader.parmRecurrenceData(sysRecurrenceData);

// Set the batch alert configurations
sampleBatchHeader.parmAlerts(NoYes::No, NoYes::Yes, NoYes::No, NoYes::No, NoYes::No);
```

Add a task to the batch job

You add tasks to the batch job by calling the *addTask* method. The first parameter for this method is an instance of a batch-executable class that is scheduled to execute as a batch task:

```
void addTask(Batchable batchTask,
 [BatchConstraintType constraintType])
```

Another way to create a task is to use the *addRuntimeTask* method, which creates a dynamic batch task. This task exists only for the current run; it is copied into the history tables and deleted at the end of the run. It copies settings such as the batch group and child dependencies from the *inheritFromTaskId* task:

```
void addRuntimeTask(Batchable batchTask,
RecId inheritFromTaskId,
[BatchConstraintType constraintType])
```

Define dependencies between tasks

The *BatchHeader* class provides the *addDependency* method, which you can use to define a dependency between the *batchTaskToRun* and *dependsOnBatchTask* tasks.

You can use the optional parameter *batchStatus* to specify the type of the dependency. By default, a dependency of type *BatchDependencyStatus::Finished* is created, which means that a task starts execution only if the task that it depends on finishes successfully. Other options are *BatchDependencyStatus::Error* (the task starts execution if the preceding task finishes with an error) and *BatchDependencyStatus::FinishedOrError* (the task starts execution if the preceding task finishes with any status result). The following example shows the signature of the *addDependency* method:

```
public BatchDependency addDependency(
    Batchable batchTaskToRun,
    Batchable dependsOnBatchTask,
   [BatchDependencyStatus batchStatus])
```

Save the batch job

The final step in creating the batch job using the Batch API is to save the job by calling the *batchHeader.save* method. The *save* method inserts records into the *BatchJob*, *Batch*, and *BatchConstraints* tables, from which the batch server can automatically pick them up for execution.

Example of a batch job

The following example shows how to create a batch job and add two batch tasks by using the Batch API. The example assumes that a batch-enabled class named *ExampleBatchTask* already exists:

```
static void ExampleSchedulingJob (Args _args)
{
    BatchHeader     sampleBatchHeader;
    RunBaseBatch    sampleBatchTask;
```

```
    // create batch header
    sampleBatchHeader = BatchHeader::construct();

    // create and add batch tasks
    sampleBatchTask1 = new ExampleBatchTask();

    sampleBatchHeader.addTask(sampleBatchTask1);

    sampleBatchTask2 = new ExampleBatchTask();

    sampleBatchHeader.addTask(sampleBatchTask2);

    // add dependencies between batch tasks
    sampleBatchHeader.addDependency(sampleBatchTask1, sampleBatchTask2);

    // save batch job in the database
    sampleBatchHeader.save();
}
```

For more examples of programmatic batch job creation, see "Walkthrough: Extending *RunBaseBatch* Class to Create and Run a Batch," at *http://msdn.microsoft.com/en-us/library/cc636647.aspx*.

Manage batch execution

The final step in implementing a batch job is to manage the execution process. Before a batch job can be executed on an AOS instance, you must configure the AOS instance as a batch server and set up the batch groups that tell the system which AOS instance should execute the job.

In addition to these initial configuration tasks, you'll likely need to manage the batch tasks and jobs: checking status, reviewing history, and sometimes canceling a batch job. You'll probably also need to debug a batch task at some point. The following sections describe how to configure an AOS instance as a batch server, set up batch groups, manage batch jobs, and debug a batch task.

Configure the batch server

You can configure an AOS instance to be a batch server, including specifying when the batch server is available for processing and how many tasks it can run, by using the Server Configuration form. The Server Configuration form is located at System Administration > Setup > System > Server Configuration. Note that the first AOS instance is automatically designated as a batch server, but you can configure additional AOS instances manually as batch servers.

Tip Use multiple batch servers to enable parallel processing and increase processing throughput.

1. In the Server Configuration form, select a server in the left pane.
2. Select the Is Batch Server check box to enable batch processing on the server, as shown in Figure 18-5.

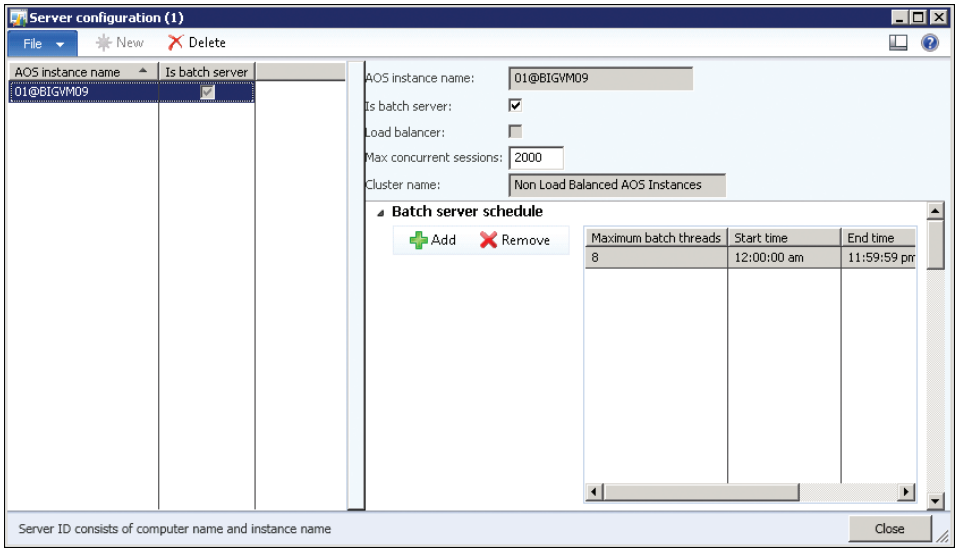


FIGURE 18-5 The Server Configuration form.

3. On the Batch Server Schedule FastTab, click Add to enter a new schedule. Enter the maximum number of batch tasks that can be run on the AOS instance at one time. The server continues to pick up tasks from the queue until it reaches its maximum.
4. Enter a starting time in Start Time and an ending time in End Time to specify the time window in which the server processes batch jobs. Press Ctrl+N to enter an additional time window.

Tip It's a good idea to exclude a server from batch processing when it is busy processing regular transactions. You can set server schedules so that each AOS instance is available for user traffic during the day and batch traffic overnight. Keep in mind that if the server is running a task when its batch processing availability ends, the task continues running to completion. However, the server doesn't pick up any more tasks from the queue.

Create a batch group

A batch group is a logical categorization of batch tasks that lets a user (typically a system administrator) determine which AOS instance runs the batch task. This section describes how to create a batch group so that it can be assigned to a specific server for execution. The first step is to create batch groups by using the Batch Group form at System Administration > Setup > Batch Group.

To create a batch group, press Ctrl+N in the Batch Group form, and then type a name and description for the batch group. The Batch Group form is shown in Figure 18-6.

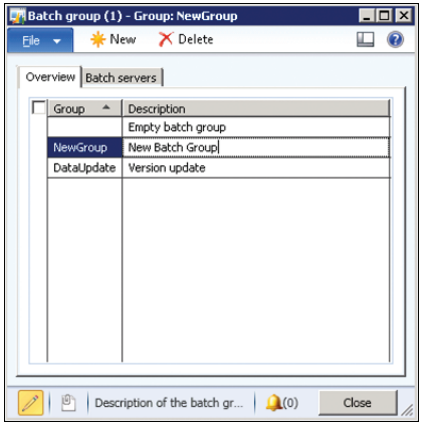


FIGURE 18-6 The Batch Group form.

Note By default, the system contains an empty batch group that can't be removed. This is a default batch group for tasks that are not explicitly assigned to a group.

After you create batch groups, assign each group to a server as follows:

1. In the Server Configuration form (shown in Figure 18-7), click the Batch Server Groups FastTab. The Selected Groups list shows the batch groups specified to run on the selected server.
2. In the Remaining Groups list, select a group, and then click the left arrow button to add this group to run on the selected server.

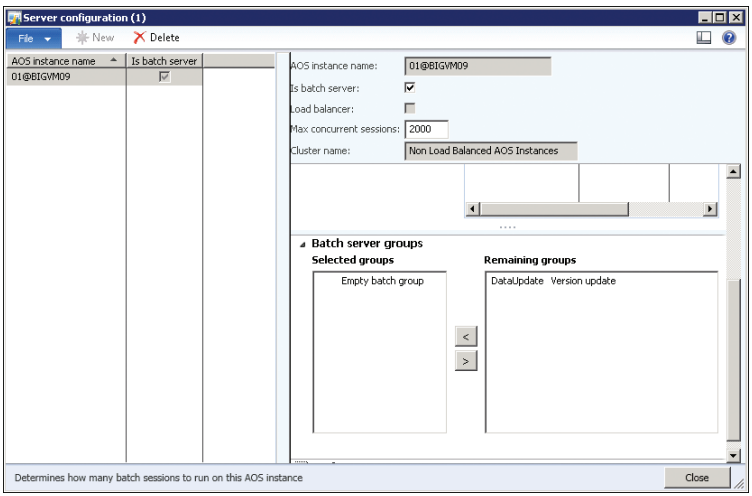


FIGURE 18-7 The Server Configuration form.

Manage batch jobs

After you create and schedule a batch job, you might want to check its status, review its history, or cancel it. The following sections describe some of the most common management tasks associated with batch jobs.

View and change the batch job status

The Batch Job list form provides a snapshot view of the current state of batch jobs. The list displays the progress and the status of running and completed jobs. It also displays any jobs that are scheduled to start soon.

You can change the status of a batch job by selecting the batch job in the list and then following these steps:

1. Click Functions, and then click Change Status.
2. In the Select New Status dialog box, select a new status for the job. For example, if the status is Waiting, you can temporarily remove the batch job from the waiting list by changing the status to Withhold.

Tip If a job exits with a status of Error/Ended and you want to rerun the job, change its status to Waiting. The job will automatically be picked up by the server for execution.

You can cancel a batch job by changing its status to Canceling. Tasks in the Waiting or Ready state are changed to Not Run; currently executing tasks are interrupted, and their status is changed to Canceled.

Control the maximum retries

If an AOS fails because of an infrastructure failure or a power outage while a batch task is executing, the batch framework has the built-in capability to retry tasks after the AOS is restarted. Any tasks that were left in an executing state, and that have not reached the maximum retry limit, are changed to the Ready state and will run shortly after the failure.

Tip If you create custom tasks and want to enable retries, design the task so that it is idempotent—that is, it can be executed multiple times without unexpected consequences.

You can modify the Maximum Retries attribute for each batch task on the General tab. By default, the value is set to 1; when the Actual Retries field on the Update tab exceeds the maximum number of retries, the batch task fails. When this happens, the recurrence that is set for the batch job is not honored, and the status of the batch job is set to either Success or Error.

Review the batch job history

You can view a history of all batch jobs that have finished running in the Batch Job History form at System Administration > Inquiries > Batch Job History. This form displays detailed information about the status of the jobs, including any messages encountered while the batch job was running.

You can also view the logs for each batch job as follows:

- To view log information for an entire batch, select a batch job, and then click Log.
- To view log information for individual tasks, select a batch job, and then click View Tasks. In the Batch History list form, select a task, and then click Log.

Tip In the batch job settings, you can specify when log information is written to the history tables: Always (the default), On Error, or Never. Use On Error or Never to save disk space for batch jobs that run constantly. This option is located on the General tab of the Batch Job form.

Debug a batch task

Because batch tasks run in noninteractive mode and X++ executes in the common language runtime (CLR), to debug a batch task, you have to perform additional steps to configure the AOS and the Visual Studio debugger, in addition to setting up breakpoints.

First, configure the AOS for batch debugging. This is necessary for two reasons. First, the AOS modifies the X++ assembly to disable Just-In-Time (JIT) optimizations in the CLR. This is necessary to enable variables and object contents to be viewed and analyzed in the debugger. Second, the AOS produces source files containing the X++ code under the server Bin\XppIL\Source folder. You can open these files in Visual Studio to set breakpoints and perform common tasks, such as step ping into and step ping over.

Configure AOS for batch debugging

Use the Microsoft Dynamics AX Server Configuration Utility (see Figure 18-8) to configure the AOS for batch debugging. The utility is available on the computer on which you installed the AOS. To do this, perform the following steps:

1. Open the Microsoft Dynamics AX Server Configuration Utility. Click Start > All Programs > Administrative Tools > Microsoft Dynamics AX 2012 Server Configuration.
2. Select the Enable Breakpoints to Debug X++ Code Running on This Server check box.
3. Click OK to close the utility, and then restart the AOS.

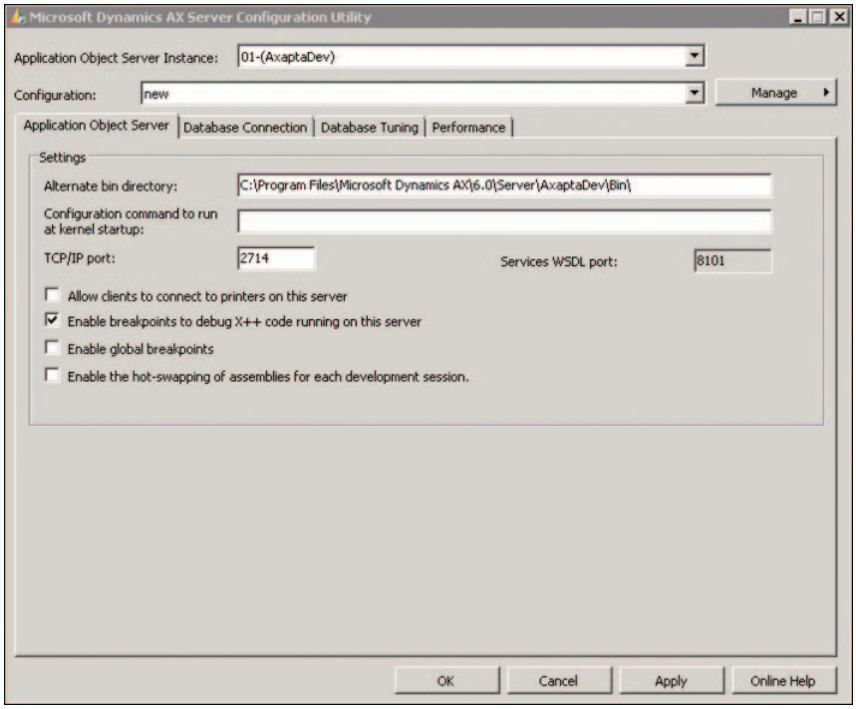


FIGURE 18-8 The Microsoft Dynamics AX 2012 Server Configuration Utility.

Configure Visual Studio for debugging X++ in a batch

To configure Visual Studio for batch debugging, attach to the AOS process Ax32Serv.exe by following these steps:

1. In Visual Studio, on the Debug menu, click Attach To Process.
2. When the Attach To Process dialog box opens (see Figure 18-9), click Select to select Managed (v4.0) code, and then select the following check boxes:
   - Show Processes From All Users
   - Show Processes In All Sessions
3. Click Ax32Serv.exe, and then click Attach.

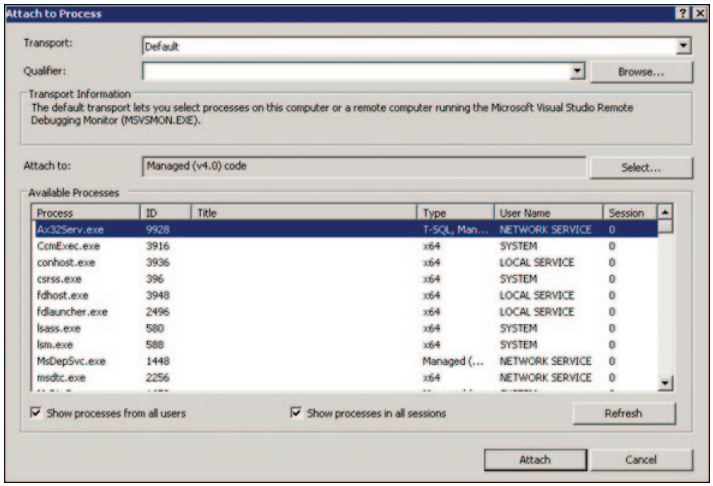


FIGURE 18-9 The Attach To Process dialog box in Visual Studio.

The next step is to disable the Just My Code option, so that the debugger breaks on X++ source code. To do this, perform the following steps:

1. On the Tools menu, click Options, and then navigate to the Debugging\General node (shown in Figure 18-10).

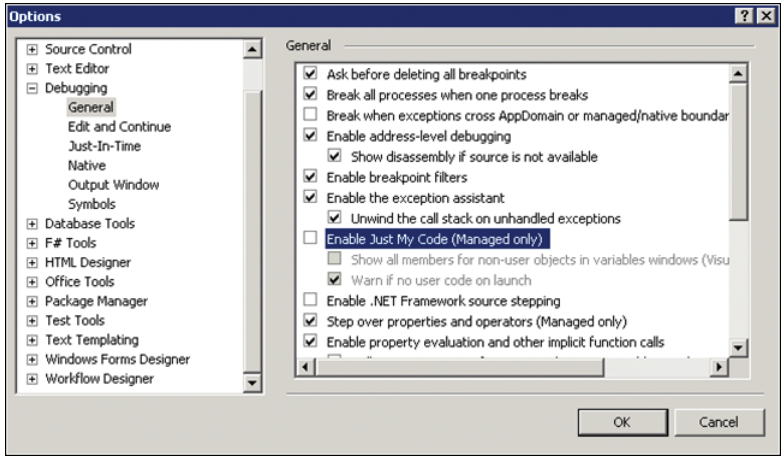


FIGURE 18-10 The Options dialog box in Visual Studio.

Clear the Enable Just My Code (Managed Only) check box, and then click OK.

After you complete these steps, you can open the X++ source code from the Bin\XppIL\Source folder on the server and set breakpoints.

CHAPTER 19

Application domain frameworks

In this chapter

Introduction 633
The organization model framework 634
The product model framework 643
The operations resource framework 648
The dimension framework 654
The accounting framework 659
The source document framework 664

Introduction

Microsoft Dynamics AX 2012 includes a number of new domain-specific application frameworks that improve code reuse between application modules, reduce the need to assign artificial relationship roles in application modules, and reduce the number of tightly coupled interdependencies between application modules. For example:

- The operations resource role that is assigned to people and work centers by the operations resource framework eliminates the need to assign employees and independent contractors artificially to the work center role, so that they can participate in production planning and scheduling activities.
- The performance dimensions extended from the dimension framework, the double-entry subledger journal provided by the accounting framework, and the source document abstraction provided by the source document framework enable operations processes such as purchasing, receiving, and invoicing to be decoupled from accounting processes such as budget control and financial reporting.

This chapter provides a short description of the conceptual foundation of some of the key application domain frameworks, in addition to links to white papers that provide more detailed implementation guidelines and code samples. This chapter does not contain an exhaustive list of application domain frameworks for Microsoft Dynamics AX 2012. Instead, it is intended to provide an overview of important new functionality and suggestions about how you can use it. For information about additional application domain frameworks, see "White Papers for Developers" at *http://technet.microsoft.com/EN-US/library/hh272882*. This page is updated frequently with links to new white papers.

Note By convention, the names of entities in the conceptual domain models in this chapter are denoted in title case and italicized for emphasis.

The organization model framework

Microsoft Dynamics AX 2012 introduces a new organization model framework. This framework is designed to model key scenarios that are required by government organizations and corporations that have global operations, and those that have separate legal and operating organization structures. The organization model framework extends the company feature that was used in Microsoft Dynamics AX 2009 and earlier versions.

With the new types of organizations that are included in Microsoft Dynamics AX 2012, you can model organizations in Microsoft Dynamics AX to mirror the way you operate them without having to customize the application. Most organizations go through an iterative operating cycle of monitor, measure, analyze, and improve. The analysis phase results in new business rules and policies and new strategic and operational initiatives to improve the organization's performance. With the organization framework, you can create hierarchical structures that enable this cycle of performance improvement.

How the organization model framework works

The organization model has two major components: *Organization Types* and *Organization Hierarchies*. The following sections describe these components.

Organization types

The organization model in Microsoft Dynamics AX 2012 introduces two new types of organizations: *Legal Entity* and *Operating Unit*.

- ***Legal Entity*** An organization with a registered or legislated legal structure that has the authority to enter into legal contracts and that is required to prepare statements that report on its performance. A legal entity and a company in Microsoft Dynamics AX 2012 are semantically the same. However, some functional areas in the application are still based on a data model that uses the concept of a company (or DataArea). These areas may have the same limitations as in Microsoft Dynamics AX 2009 and may have an implicit data security boundary.
- ***Operating Unit*** An organization that divides the control of economic resources and operational processes among people who have a duty to maximize the use of resources, to improve processes, and to account for their performance.

Microsoft Dynamics AX 2012 includes several types of operating units:

- **Business unit** A semi-autonomous *Operating Unit* that is created to meet strategic business objectives.
- **Cost center** A type of *Operating Unit* that describes an organization that is used to track costs or expenses. A cost center is a cost accumulator, and it is used to manage costs.
- **Department** A type of *Operating Unit* that may have profit and loss responsibility and might consist of a group of cost centers. Departments also are often created based on functional responsibility or skill, such as sales and marketing.
- **Value Stream** A type of *Operating Unit* that is commonly used in lean manufacturing. In lean manufacturing, a value stream owns one or more production flows that describe the activities and flows needed to supply a product, an item, or a service to the consumers of the product.
- **Retail Channel** A type of *Operating Unit* that is commonly used in retail to represent a retail channel.

A *Team* is also a type of *Internal Organization*, but it is an informal group of people that is typically created for a specific purpose over a short duration. Teams might be created for specific projects or services. The other types of organizational units described here are more permanent, although they could require frequent minor updates or major changes because of restructuring.

These types of operating units support application functionality in Microsoft Dynamics AX 2012. However, every industry and business has unique requirements for its operating units and may call them by different names. Additionally, organizations can create custom types of operating units to meet their needs. For more information, see the section "Create a custom operating unit type," later in this chapter.

When you arrange legal entities and operating units into hierarchies and use them for aggregated reporting, to secure access to data, and to implement business policies, they help enable internal control of your organization.

Organization hierarchies

An *Organization* is a group of people who work together to perform operational and administration processes. *Organization Hierarchies* represent the relationships between the *Organizations* that make up an enterprise or a government entity. In previous releases, companies could not be organized into a hierarchy to represent the structure of an organization. In reality, organizations typically have a hierarchical structure for the reasons mentioned in the previous section.

The organization model framework in Microsoft Dynamics AX 2012 supports the creation of multiple hierarchies that take effect on multiple dates. This is useful in restructuring scenarios, where you want the updated hierarchy to become effective at a certain date in the future. The framework also supports hierarchies that are used for multiple purposes.

A *Purpose* defines how the *Organizational Hierarchy* is used in application scenarios. The *Purpose* that you select determines the types of organizations that can be included in the hierarchy. Table 19-1 shows the types of organizations that you can use for each hierarchy purpose that is included in Microsoft Dynamics AX 2012.

TABLE 19-1 Purposes and organization types.

Purpose	Description	Organization types
Procurement internal control	Use this purpose to define policies that control the purchasing process.	All
Expenditure internal control	Use this purpose to define policies for expense reports.	All
Organization chart	Use this purpose in human resources to define reporting relationships.	All
Signature authority internal control	Use this purpose to define policies for signing limits. These policies control the spending and approval limits that are assigned to employees.	All
Vendor payment internal control	Use this purpose to define policies for the payment of vendor invoices.	Legal entities
Audit internal control	Use this purpose to define policies for identifying documents for audit.	Legal entities
Centralized payments	Use this purpose to make payments by one legal entity on behalf of other legal entities.	Legal entities
Security	Use this purpose to define the data security access for organizations.	All
Retail assortment	Use this purpose to define assortments for retail channels.	All
Retail replenishment	Use this purpose to define replenishment configurations.	All
Retail reporting	Use this purpose to define dimensions for retail reporting cubes.	All

The organization model has a significant impact on the implementation of Microsoft Dynamics AX 2012 and on the business processes being implemented. Executives and senior managers from different functional areas such as finance and accounting, human resources, operations, and sales and marketing should participate in defining the organization structures.

Figure 19-1 shows the conceptual domain model of the organization model framework.

When to use the organization model framework

You use the organization model framework to model how the business operates. You can use the organization model framework in two ways: by using the built-in integration with other application frameworks and existing Microsoft Dynamics AX modules, or by modeling custom scenarios to meet the needs of your organization.

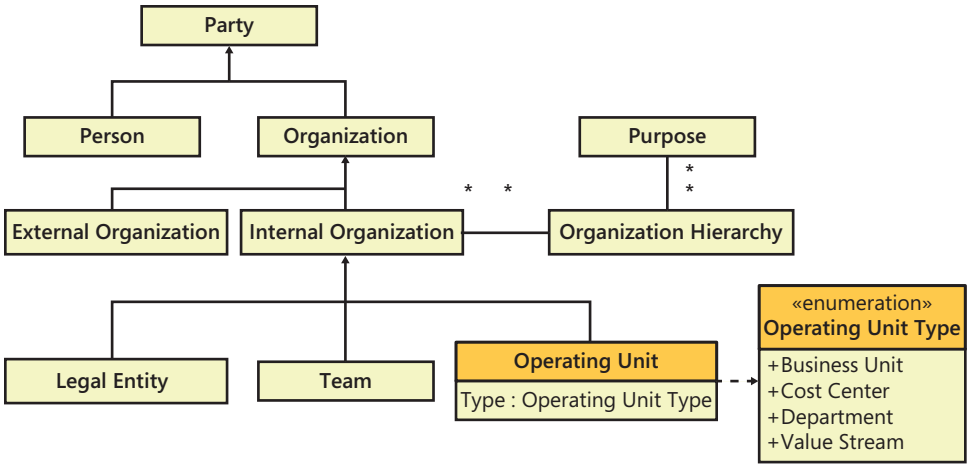


FIGURE 19-1 The organization model framework.

Integration with other frameworks application modules

The organization model framework is inherently integrated with certain frameworks and modules in Microsoft Dynamics AX:

- **Address book** All internal organizations—legal entity, operating unit, and team—are types of the *Party* entity. This means that these organizations can use the capabilities of the address book to store address and contact information. For more information, see the white paper, "Implementing the Global Address Book Framework" at *http://technet.microsoft.com/en-us/library/hh272867.aspx*.
- **Financial dimensions** You can use legal entities and operating units to define financial dimensions and then use those financial dimensions in account structures. By using organizations as financial dimensions, an enterprise or government entity can analyze an organization's financial performance. If two types of organizations are used as separate financial dimensions in the account structure, the relationships between organizations described through hierarchies can also be used as constraints. For more information, see the section "The dimension framework" later in this chapter.
- **Policy framework** You can use the policy framework to define an internal control policy for an organization. The policy framework can be used to define policies for expense reports, purchase requisitions, audit control of documents, and vendor invoice payments. The policy framework provides support for override and default behavior for organizations based on their hierarchies, and enables internal management control of organizations to facilitate cost control, fraud detection, better operating efficiency, and better performance in general. For more information, see the white paper, "Using the Policy Framework," at *http://technet.microsoft.com/en-us/library/hh272869.aspx*.

- **Extensible data security** The new extensible data security framework provides capabilities to secure data based on any condition. Security access to organizations can be defined based on hierarchies. For more information, see Chapter 11, "Security, licensing, and configuration."

The organization model framework is also used in the following application modules:

- **Procurement and sourcing** The lines of a purchase requisition are created for a buying legal entity, and they are received by an operating unit, such as a cost center or a department. The organization model framework enables various scenarios by allowing the viewing or creation of purchase requisitions for anyone buying legal entities and receiving operating units in which you have access to create purchase requisitions.
- **Human resources** In human resources, workers hold employment contracts in a legal entity and have a position in a department. All transaction scenarios in human resources use these concepts to view and modify data.
- **Travel and expense** Expense reports and expense line items are associated with a legal entity to which the expense line item should be charged from a statutory perspective, and they also are associated with an operating unit for internal reporting.

Model your own functional scenarios

You can use the organization model framework to model your own scenarios. For example, a common scenario for data security is to filter application data based on a user's roles and membership in internal organizations. For example, organizations might seek to limit an individual account manager's access to specific sales orders based on geography, allowing her to view only the sales orders that originate in her region.

You can set up a new scenario or customize an existing scenario by taking the following high-level steps:

1. Define or change the data model.
2. Create a new table with the *SaveDataPerCompany* property set to *No*. If you are working with existing tables that are marked per-company, change the value of the *SaveDataPerCompany* property from *Yes* to *No*.
3. Reference organizations as foreign keys (FKs) on the table. It may be necessary to reference an operating unit and a legal entity if the legal entity cannot be established through legal entity or operating unit organization hierarchies.
4. If the table includes redundant data in the *Legal Entity* field, set up hierarchical constraints between legal entities and operating units to maintain data consistency.
5. Build a new form (for example, a list page) for the scenarios, or change the existing user experience to view or maintain data. You can use custom filters to make it possible for users to view and maintain data across organizations.

6. Apply default organizations on the table in financial dimensions by including them in account structures.
7. Create extensible data security policies that are based on the organizations that the user belongs to or has access to.
8. Use the policy framework to set up policies to apply when users access data in the scenario.

Extending the organization model framework

You can extend the organization model framework by creating a custom type of operating unit or a custom purpose, or by extending the hierarchy designer, a tool that is included with Microsoft Dynamics AX 2012.

Create a custom operating unit type

A core extensibility scenario is to extend the organization model to accommodate specific vertical industry requirements. For example, branches, schools, and school districts are essentially organization concepts, and you can model them as new types of operating units.

Suppose that you want to create an operating unit called *Branch*. To do so, you would follow these steps:

1. Create a new base enum value for the new *Operating Unit Type*.
2. Create a view.
3. Create a menu item for the new operating unit.

The following sections describe these steps in more detail.

Create a new base enum value Define a new base enum value for the *OMOperatingUnitType* enum, which corresponds to the new type of operating unit:

1. In the Application Object Tree (AOT), navigate to Data Dictionary\Base Enums\ OMOperatingUnitType.
2. Right-click the *OMOperatingUnitType* enum, click New Element, and then add an element named *Branch*.
3. In the Properties window for the new element, set both the *Name* and *Label* properties to *Branch*. Change the default *EnumValue* property to an integer value that will prevent clashes between your new enum and future enums added by the Microsoft Dynamics AX team.

Create a view Define a view, DimAttributeBranchView, which is similar to views created for other types of operating units. For example, the view DimAttributeOMBusinessUnit was created for business units. The DimAttributeOMBusinessUnit view contains the fields that allow the OMOperatingUnit table to be used as a financial dimension for all business units specified.

1. In the AOT, navigate to the Data Dictionary\Views node, and then locate DimAttributeOMBusinessUnit.
2. Duplicate DimAttributeOMBusinessUnit: Right-click the *DimAttributeOMBusinessUnit* node, and then click Duplicate. The AOT will create a node called *CopyOfDimAttributeOMBusinessUnit*.
3. In the Properties window for the newly created view, set the properties as shown in the following table:

Property	Value
Name	*DimAttributeBranchView*
Label	*Branches*
SingularLabel	*Branch*
DeveloperDocumentation	Type a description for the view. For example, The *DimAttributeBranchView* contains all records from the OMOperatingUnit table that are specified as branches.

4. Under the new DimAttributeBranchView view, locate the OMOperatingUnitTypeOMBusinessUnit range. The range is under the Metadata\Data Sources\BackingEntity(OMOperatingUnit)\Ranges node.

In the Properties window for the OMOperatingUnitTypeOMBusinessUnit range, set the properties as shown in the following table:

Property	Value
Name	*OperatingUnitTypeBranch*
Field	*OMOperatingUnitType*
Value	*Branch*

Create a menu item Finally, create a menu item for the new operating unit type as follows:

1. Under Menu Item\Display, create a menu item named *BranchMenuItem*.
2. In the Properties window for *BranchMenuItem*, set the following properties:

Property	Value
Name	*BranchMenuItem*
Label	*Branches*
Object	*OMOperatingUnit*
EnumTypeParameter	*OMOperatingUnitType*
EnumParameter	*Branch*

The new operating unit type will now appear in the list of operating unit types that are available when a system administrator creates a new operating unit.

Create a custom purpose

You can extend the Microsoft Dynamics AX organization model to create a custom purpose. A purpose defines how the organization hierarchy is used in application scenarios.

Suppose you want to create a new purpose called Sales. To do so, you would follow these steps:

1. Create a new base enum value for the new purpose.
2. Create a method to add the new purpose, and then call that method to add the purpose to the HierarchyPurposeTable table.

The following sections describe these steps in more detail.

Create a new base enum value First, create a new base enum value for the new purpose. To do so, you follow these steps:

1. In the AOT, navigate to Data Dictionary\Base Enums\HierarchyPurpose.
2. Right-click the *HierarchyPurpose* enum, click New Element, and then add an element named *Sales*.
3. In the Properties window for the new element, set the Name and Label properties to *Sales*. Change the default EnumValue to an integer value that will prevent clashes between your new enum and future enums that are added by Microsoft or independent software vendors (ISVs).

Create and call a method to add the new purpose Next, create the method to add the new purpose, and then call the method to add it to the table.

1. In the *Classes* node in the AOT, locate *OMHierarchyPurposeTableClass*.
2. Duplicate the *addSecurityPurpose* method: Right-click the *addSecurityPurpose* node, and then click Duplicate. The AOT will create a method called *CopyOfaddSecurityPurpose*.
3. Replace the code for the *CopyOfaddSecurityPurpose* method with the following code. This code renames the method:

```
private static void addSalesPurpose()
{
    OMHierPurposeOrgTypeMap omHPOTP;
    select RecId from omHPOTP
        where omHPOTP.HierarchyPurpose == HierarchyPurpose::Sales;
    if (omHPOTP.RecId <= 0)
    {
        omHPOTP.clear();
        omHPOTP.HierarchyPurpose = HierarchyPurpose::Sales;
        omHPOTP.OperatingUnitType = OMOperatingUnitType::OMAnyOU;
        omHPOTP.IsLegalEntityAllowed = NoYes::No;
        omHPOTP.write();
        omHPOTP.clear();
        omHPOTP.HierarchyPurpose = HierarchyPurpose::Sales;
        omHPOTP.OperatingUnitType = 0;
        omHPOTP.IsLegalEntityAllowed = NoYes::Yes;
        omHPOTP.write();
    }
}
```

The preceding code is similar to the code in most of the methods of the *OMHierarchyPurposeTableClass* class. The code was changed in only the places where the *Hierarchy Purpose* enum values are referenced. In the code, you can see three occurrences of HierarchyPurpose::Sales.

4. In the *OMHierarchyPurposeTableClass* class, update the *populateHierarchyPurposeTable* method to call the new method that you created, by adding the following line of code:

```
OMHierarchyPurposeTableClass::addSalesPurpose();
```

The following code shows the modification to the *populateHierarchyPurposeTable* method:

```
public static void populateHierarchyPurposeTable()
{
    OMHierPurposeOrgTypeMap omHPOTP;
    if (omHPOTP.RecId <= 0)
    {
        ttsbegin;
        OMHierarchyPurposeTableClass::AddOrganizationChartPurpose();
        OMHierarchyPurposeTableClass::AddInvoiceControlPurpose();
        OMHierarchyPurposeTableClass::AddExpenseControlPurpose();
        OMHierarchyPurposeTableClass::AddPurchaseControlPurpose();
        OMHierarchyPurposeTableClass::AddSigningLimitsPurpose();
        OMHierarchyPurposeTableClass::AddAuditInternalControlPurpose();
        OMHierarchyPurposeTableClass::AddCentralizedPaymentPurpose();
        OMHierarchyPurposeTableClass::addSecurityPurpose();
            //Add the following line.
        OMHierarchyPurposeTableClass::addSalesPurpose();
        ttscommit;
    }
}
```

After you complete these steps, the new purpose will appear under Organization Administration > Setup > Organization > Organization Hierarchy Purposes.

Extend the hierarchy designer

System administrators can view or modify organizational hierarchies by using the hierarchy designer. This form is available through Organization Administration > Setup > Organization > Organization Hierarchies.

Developers have a few options for extending the hierarchy designer. The hierarchy designer control can be customized for four parameters of the organization nodes within the hierarchy: border color, node image, top gradient color, and bottom gradient color. For more information, download the white paper "Implementing and Extending the Organization Model in Microsoft Dynamics AX 2012" from *http://download.microsoft.com/download/4/E/3/4E36B655-568E-4D4A-B161-152B28BAAF30/Implementing_and_extending_the_organization_model_in_Microsoft_Dynamics_AX_2012.pdf*.

The product model framework

Microsoft Dynamics AX 2012 offers a flexible product data management framework, supporting both centralized and legal-entity-specific management of information about a product, which is defined as an item or a service that results from an economic activity.

How the product model framework works

In the product model, product information is centralized around the concept of a *Product*, which represents information that is shared across the organizational structure, and the concept of a *Released Product*, which controls information that is specific to a legal entity. Figure 19-2 shows the conceptual domain model of the product model framework.

Product types and subtypes

A *Product* can be an *Item*—which represents a physical entity for which inventory levels can be tracked, like finished goods or raw components—or a *Service*—modeling a nonphysical entity for which inventory levels are not tracked, like providing consulting services. A *Product* can be divided further into additional subtypes: a *Distinct Product* that is uniquely identifiable and can be used in economic activities, or a *Product Master*. A *Product Master* is a standard or functional product representation that serves a basis for configuring distinct *Product Variants*. In this case, a *Product Variant* is a uniquely identifiable *Product* that can be used in economic activities. However, because all *Product Variants* are bound to a *Product Master*, they share a significant part of the information defined for the *Product Master*.

Consider a manufacturing company named Contoso that sells different models of bicycles, which come in different colors and sizes, and accessories such as bicycle lights. In this scenario, every bicycle model can be modeled as a *Product Master*, with each color and size combination defining a *Product Variant*. Lights do not come in various configurations, so they can be modeled as distinct *Products*.

Product dimensions

A *Product Variant* is defined by using *Product Dimensions*, which are active for a given *Product Master*. *Product Dimensions* are characteristics that uniquely identify a product. Microsoft Dynamics AX 2012 includes four *Product Dimensions*: Configuration, Size, Color, and Style. (You can rename them.) A *Product Dimension Group*, which is required for a *Product Master*, encompasses the information about the *Product Dimensions* that are active and can be used for controlling which *Product Dimensions* should be considered in the price calculation in trade agreements. Because *Product Dimensions* define unique *Products*, a *Product Dimension Group* must be assigned to the *Product Master* and is shared across the organizational structure.

For example, Contoso sells two bicycle models. Each model is available in three sizes (small, medium, and large) and three colors (black, red, and white). Contoso can model this assortment by creating a *Product Dimension Group* named Bicycles, with Size and Color as the active *Dimensions*, and assigning that *Product Dimension Group* to the two *Product Masters* that represent the two bicycle models. Because Contoso also sells helmets, which come in one color but in different sizes, Contoso can create a second *Product Dimension Group* called *Helmets* that has only one active *Dimension*: Size.

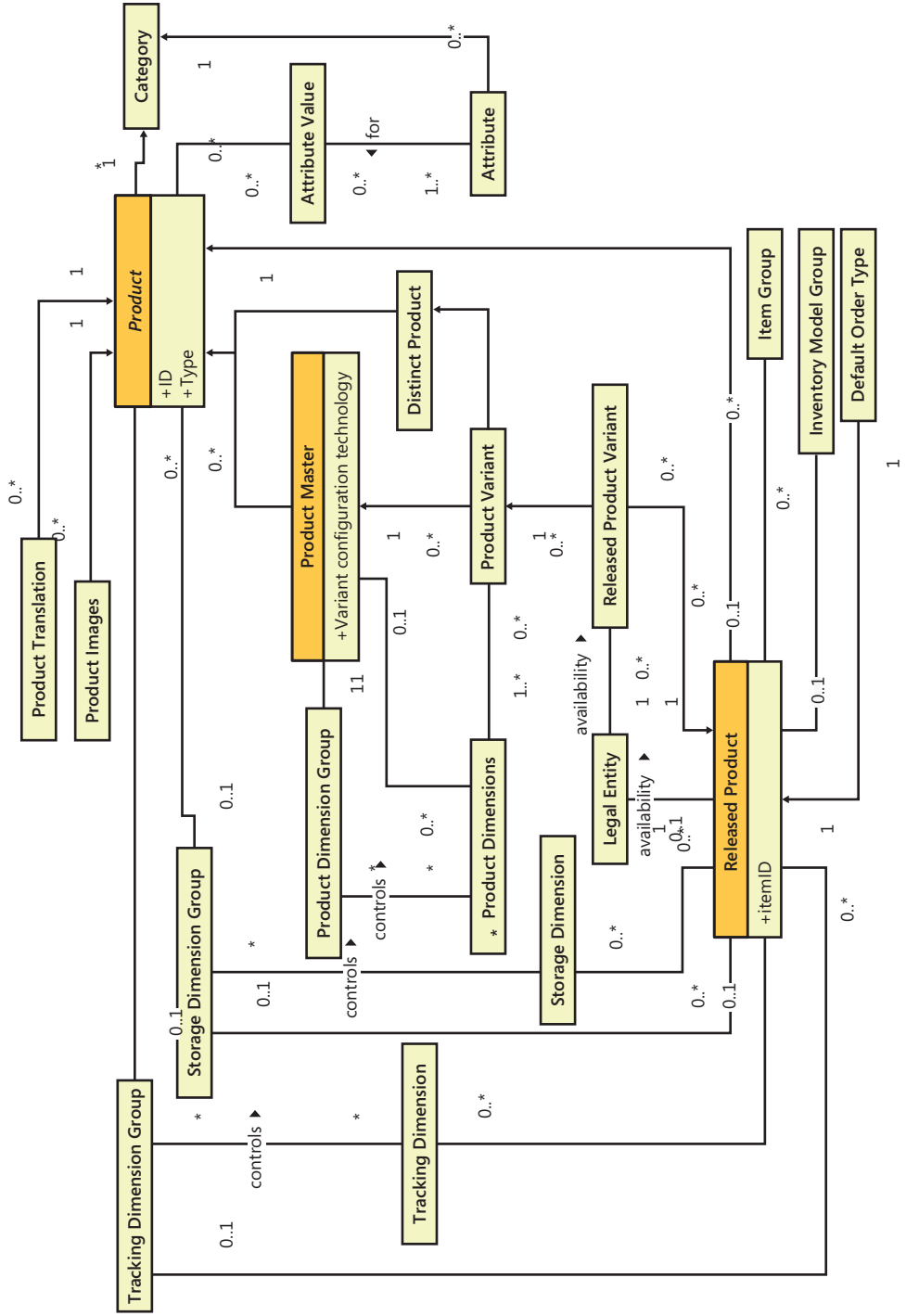


FIGURE 19-2 The product model framework.

Storage and tracking dimensions

Besides *Product Dimensions*, there are two more types of dimensions: *Storage* and *Tracking*. The *Storage Dimension Group* defines the level of detail used to identify the physical location of goods, with the possible dimensions being Site, Warehouse, Location, and Pallet. The *Tracking Dimension Group* defines how specific instances of the same *Released Product* are tracked, with *Batch* and *Serial Number* being available. These dimension groups also define policies regarding how specific dimension values affect business activities. Because these policies may differ in different legal entities, different storage and tracking dimensions can be assigned to the same product in different legal entities.

For example, Contoso might be required by law to track batch numbers of bicycle brakes in certain countries/regions, so that an entire batch can be recalled easily in case a widespread defect is discovered that causes a safety hazard. This requirement does not apply in other countries/regions, so Contoso may choose not to track brake batches in these countries/regions.

Released products

All of the information described so far applies to the concept of a *Product*, representing the information that is shared across the entire organization structure. But this information is not sufficient for a legal entity to be able to use a product. To enable a legal entity to use a product, the product, together with the relevant product variants, has to be released to the legal entity. On a *Released Product*, you can define various policies that control how these products can be used within that legal entity. These policies include storage and tracking dimension groups (unless they were defined on the shared product information), *Inventory Model Group* and *Item Group*, and various inventory and costing policies that apply to the product. Once these policies are defined, the product can be used for transactions within the legal entity.

Product availability within legal entities can be controlled both on the product and the product variant level. For example, a specific bicycle model can be released only to Contoso U.S. because the company wants to sell it only in that market. Another model can be released to Contoso U.S. in only black and red because the company has decided that it's not worth investing in white bikes in the United States.

Additional product information

Beside the required information, additional details can be provided for a product, either to provide a reference or to control the interaction with other system components. For example, you might define *Product Translations* for a product to control the product name or descriptions that are displayed in various contexts. Similarly, you can attach *Images* to provide a graphical representation of the product. An example of the information used to control other components may be the *Default Order Type*, which a legal entity can set to decide the type of order that is generated to cover the demand for a particular product. For example, Contoso Italy, which owns a clothing production facility, could set the *Default Order Type* for cycling clothes to *Production*, while bicycles, which are bought from external vendors, would have the default order type set to *Purchase*.

Product attributes and categories

Microsoft Dynamics AX 2012 introduces the concept of product *Attributes*. User-defined product *Attributes* can be associated with a product *Category* to describe characteristics that are common to all products within that *Category*. Product attributes can be of various data types. Each *Attribute* might have a default value within that *Category*. Once a product is added to a category, the product inherits all product attributes within that category along with their default values. However, the value of a product attribute can be changed for each product.

For example, a company in the food industry has a product attribute called *Fat* that is associated with the procurement category for milk products. The default value is *1%*. The category contains the product *Light-milk*, which inherits the default value of *1%* from the category. The category also contains the product *Fat-milk*, which also inherits the *Fat* value. However, the value for *Fat-milk* has been overwritten with a value of *3%*. A purchasing clerk could search all products that match specific criteria; for example, to find all milk products with a *Fat* value between *1%* and *3%*.

Note Product attributes primarily provide an additional product description that can be used for search functions. You cannot use product attributes to track inventory. The system uses only *Product*, *Tracking,* and *Storage* dimensions to track physical and financial inventory.

Variant configuration technology

The product master definition has a mandatory variant configuration technology, which you use to define the configuration strategy of the *Product Master*. The configuration strategy identifies the method used to create a new *Product Variant* to meet a customer's needs. As a result of product configuration, a new product variant is created in the shared product repository and automatically released to the legal entity when product configuration takes place.

For example, in a configure-to-order environment, a manufacturing company provides a number of predefined product models with different constraints. If the existing models do not meet the expectations of a specific customer, the system creates a new unique product variant to represent the product variation that the customer wants.

Constraint-based configuration technology With the new *Product Configuration* module in Microsoft Dynamics AX 2012, you can describe a shared product model in terms of product structure, product components, attributes, and constraints. After you create a product model, you can associate it with the product master definition, which follows a constraint-based configuration strategy. This advanced configuration technology allows modeling of complex product structures.

For example, a company produces a home theater system that contains 10 components with a number of predefined constraints based on various component characteristics (attributes). During sales order entry, the system exposes a rich product configuration experience that guides the user through the configuration process to create the correct product variant successfully.

Dimension-based configuration technology In Microsoft Dynamics AX 2012, you can set up the configuration group and configuration rules as part of the general Inventory management setup. This information can be used in a bill of material (BOM) definition. You can use the configuration rules to predefine the product configurations to use as part of a specific BOM configuration.

For example, a manufacturing company produces gaming devices in several configurations. To produce these products, the company uses a BOM, which contains a raw component that comes in different configurations. With configuration groups and configuration rules, the system can enforce that only a specific raw component configuration can be included in a specific configuration of a gaming device.

This functionality allows a lightweight approach to configuring products on the order line that have relatively simple BOM structures.

Predefined variant configuration technology Predefining product variants is the simplest configuration strategy in Microsoft Dynamics AX 2012. The different product variants can be created automatically or manually based on the product dimension values, which are associated with the product master.

For example, a product master might have active dimensions of Size and Color. Every time a user adds a new color or size value, the system automatically creates all possible product variants. This functionality is especially valuable in the retail industry, where companies offer a wide range of products based on different styles, colors, and sizes.

When to use the product model framework

You can extend the product model to align the stored product information and product behavior with organizational master data management practices. These practices might include processes such as data governance, centralized master data control, or a product-specific policy. Such a policy can affect the product lifecycle or specific behavior within business processes, such as procurement, production, reserve logistics, and so on.

For example, a multi-country/region retail organization might require a process for defining an organization-wide policy such as product sales price. After the new sales price is set, that price should be used in all retail stores.

Extending the product model framework

In Microsoft Dynamics AX 2012, you can customize the product model by extending tables and classes with the prefix *EcoRes*. For example, you could begin to implement the sales price policy in the previous example by adding a new field to the EcoResProduct table to represent the sales price in the currency assigned as the primary currency in the legal entity's ledger configuration. This field is inherited automatically by all product subtypes such as distinct product, product master, and product variant, which means that you can define a specific sales price for product variations. Once the policy has been implemented, you need to expose the information in the user interface of the Product information management module to allow users to manage sales prices.

The next step is to adjust the product release process to propagate the sales price to released products. You can do this by modifying the *EcoResProductReleaseManager* class, which is responsible for the creation of released products. Specifically, you need to set a proper value in the InventTableModule table, which stores the default sales, purchase, and production prices for the released product within a legal entity.

If the shared product sales price changes, the new sales price value should be propagated to all legal entities in which the product has been released. One of the ways to achieve this is to add such logic to the *update* method of the EcoResProduct table.

The potential conceptual model is illustrated in Figure 19-3.

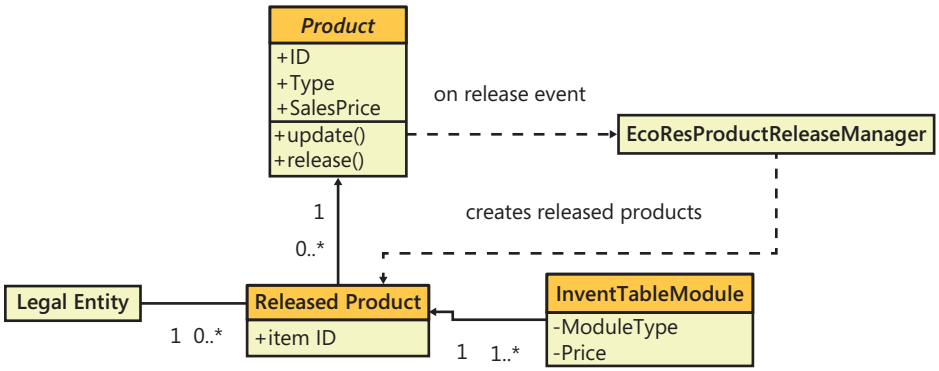


FIGURE 19-3 The customization model.

For more information about the product model framework, download the white paper "Implementing the Item-Product Data Management Framework" from *http://technet.microsoft.com/EN-US/library/hh272877*.

The operations resource framework

Designing a manufacturing process within an enterprise resource planning (ERP) system has traditionally been done by describing what activity should be performed and who should do it. This requires the process engineer to know not only how a product is built, but also which resources are available for building the product. In Microsoft Dynamics AX 2012, a new model has been put in place that allows decoupling of the process from the resources, so that the process can be described without having to reference specific resources.

How the operations resource framework works

The primary entity of the operations resource model is the *Resource*, which is defined as anything that is used for the creation, production, or delivery of an item or service other than the materials that are consumed in the process. There are multiple types of resources: Tool, Machine, Human Resource, Location, and Vendor.

A *Resource* can be a member of a *Resource Group* and the *Resource Group Membership* can change over time. You can think of a *Resource Group* as a vehicle for organizing *Resources*. A *Resource Group* is located at a particular site. A *Resource* can be a member of only a single *Resource Group* at a time. A *Resource* does not have to belong to a *Resource Group*, but the *Resource* is considered for scheduling only during the period or periods that it is connected to a *Resource Group*.

Figure 19-4 shows the conceptual domain model for *Resources* and *Resource Groups*.

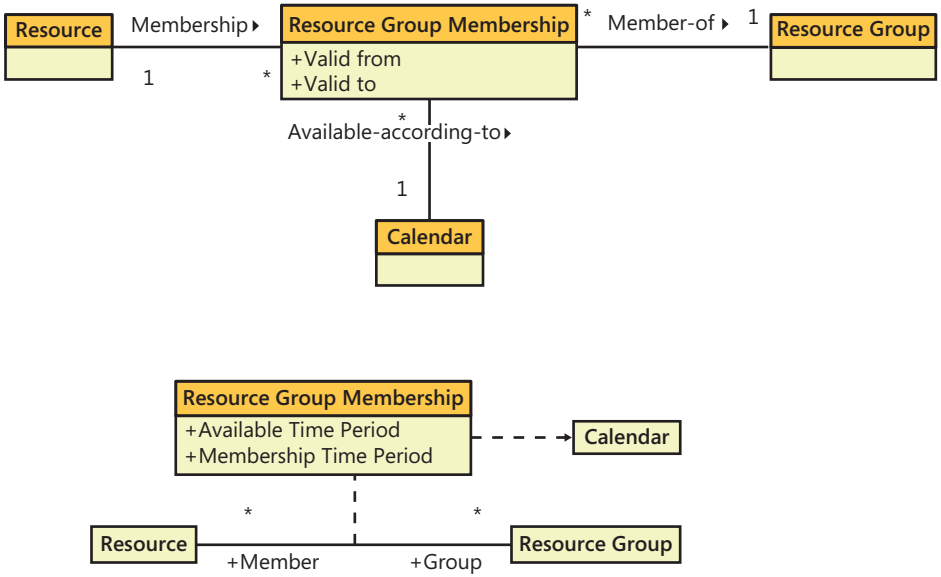


FIGURE 19-4 *Resources* and *Resource Groups*.

Capabilities

A *Capability* is the ability for a resource to perform a given activity, such as welding, pressing, or floor sweeping. A Resource can be assigned one or more *Capabilities* and can have multiple *Capabilities* on the same date. For each assignment, you can set a priority and level at which the *Capability* can be performed; for example, stamping with 4 tons of pressure.

Figure 19-5 shows the conceptual domain model for *Resources* and *Capabilities*.

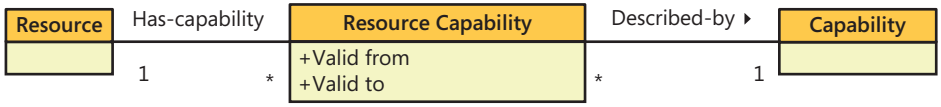


FIGURE 19-5 *Resources* and *Capabilities*.

A *Capability* can be assigned to any *Resource* regardless of its type. If a *Resource* is of the type *Human Resource,* which is associated with a worker, skills, courses, certificates, and title information from the *Human Resources* module, you can use that information in addition to the *Capability* to define the competencies of the resource.

Activities and requirements

An *Activity* is a common abstraction of the unit of work to be performed by one or more *Resources*. The *Activity* entity in itself is not visible to the user but is used internally in Microsoft Dynamics AX for a common representation of the following business entities: *Hour Forecast* (Project), *Production Route*, *Operation Relation*, *Product Model Operation Relation*, and *Product Builder Operation Relation*.

Figure 19-6 shows the conceptual domain model for an *Activity*.

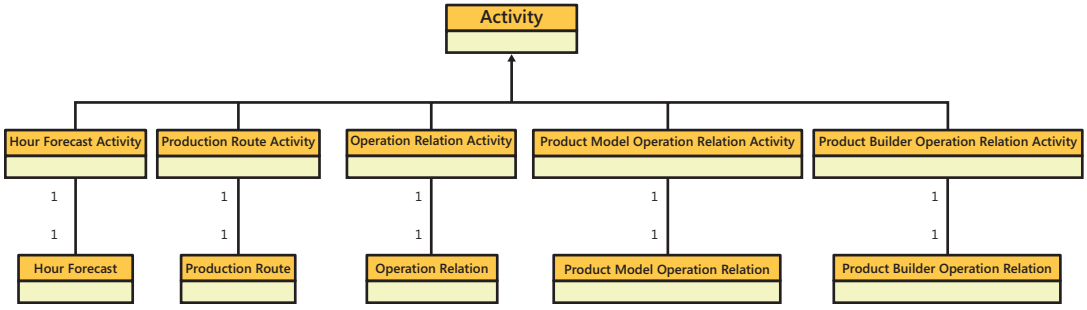


FIGURE 19-6 *Activity* model.

Each *Activity* can have a set of *Activity Requirements* that specifies how many *Resources* are needed for the *Activity* and what abilities the *Resources* must have to participate in the *Activity*. Multiple *Activity Requirements* can be contained in an *Activity Requirement Set*. In order for a *Resource* to be applicable to an *Activity*, the *Resource* must meet all of the requirements in the *Activity Requirement Set*. For each *Activity Requirement*, you can specify whether the *Requirement* should be considered when operations scheduling or job scheduling is performed. Figure 19-7 shows the conceptual domain model for *Activities, Activity Requirements Sets*, and *Activity Requirements*.

Identify applicable resources

Finding the *Resources* that are applicable for an *Activity* requires that at least the following information is known:

- The date as of which to perform the search, because *Resource* and membership information can vary over time an as-of date must be provided.
- The site context, because in most cases the site will be a limiting factor as the resources must be a member of a resource group on the site where the production takes place.
- The scheduling method, because an *Activity Requirement* can be applicable for either operations scheduling, job scheduling, or both.

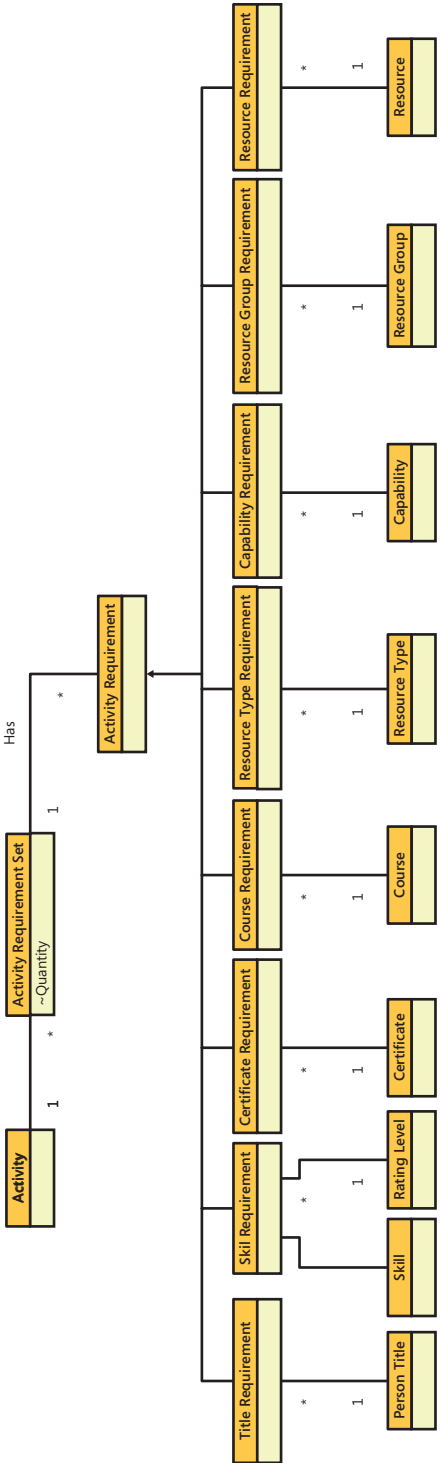


FIGURE 19-7 *Activities, Activity Requirement Sets,* and *Activity Requirements*.

Conceptually, identifying an applicable *Resource* is easy; it is simply a matter of traversing through all *Resources* while checking to determine whether the skills, capabilities, resource type, and so on, of the *Resource* match the ones stated in the requirements and ensuring that the *Resource* is not associated with a *Resource Group* that is marked as a lean work cell.

In code, you can find applicable *Resources* for an *Activity* by using one of the two main application programming interfaces (APIs) offered for the activity requirement set:

- ***applicableResourcesList*** returns the IDs of all applicable resources in a simple list.
- ***applicableResourcesQuery*** creates a query object with the WrkCtrTable table as the primary data source.

When the *Activity* has been planned by the scheduling engine, the chosen *Resource* (or *Resources*) can be found through querying the capacity reservations.

When to use the operations resource framework

You can use the operations resource framework for any activity that requires one or more resources. The framework provides good integration with the scheduling engine, which can perform the resource selection and allocate time according to the requirements and priorities.

Extensions to the operations resource framework

You can extend the operations resource framework in at least two ways: by adding a new class of activity and by adding a new class of activity requirement. The following sections provide details about the integration points and describe some of the considerations that must be taken.

Add a new class of activity

To add a new class of activity, you first need a primary table (called X in the example) that contains the activity definition, including the task to be performed, the duration, links to other activities, and so on. To connect this table to the generic activity, create a new WrkCtrXActivity table with a *0-1* relation to the WrkCtrActivity table and a *1-1* relation to the X table. Having this data structure in place makes it possible to create a form where users can fill in the activity requirements for your X table and navigate further to see the resulting applicable resources. The ProdRoute form is a good example of how such a form can be constructed and the logic that is needed to control the WrkCtrActivityRequirement and related data sources.

If the activity must be scheduled, this can be done by using the same resource scheduling engine that is used for master planning, production orders, and projects. The main class is *WrkCtrScheduler*.

Each type of activity has its own derivative class, such as *WrkCtrScheduler_Proj*, which has information about how that type of activity is handled. At the minimum, the following methods must be implemented:

- ***loadData*** Feeds the engine with information about which activities should be scheduled, the duration for applicable resources, dependencies, and links between activities, and so on.
- ***saveData*** Iterates through the results from the core engine saving the from and to time on the activity and creates capacity reservations in the WrkCtrCapRes table. It is recommended that you add a new value to the *WrkCtrCapRefType* enum and use this value when saving the capacity reservations. Doing so allows for better traceability of who owns the reservation.

Add a new class of activity requirement

If information exists that is related either directly to an operations resource or to the vendor that is associated with the resource, and that information determines the resources' ability to perform an activity, you can incorporate this information into the resource selection process.

If the information related to the resource is stored in a table named *Y*, first create a new value in the *WrkCtrActivityRequirementType* enumeration to represent the *Y* entity. Next add a new table named WrkCtrActivityYRequirement that contains a foreign key to the WrkCtrActivity table and the Y table. Because much of the logic surrounding resource requirements relies on reflection, the new table must implement a certain set of methods. Use the WrkCtrActivityPersonTitleRequirement table as an example of the table methods needed.

After the requirement table is in place, the application must be modified in several places to take the new table into consideration. The best way to ensure that the new requirement is implemented throughout the application is to use cross references for one of the existing tables, such as WrkCtrActivityPersonTitleRequirement, and then add the new table in a similar way.

For performance reasons, the matching of the resource requirements for an activity against the actual abilities of a resource is done by the core scheduling engine by converting capabilities, skills, certificates, and so on to a common property that can be compared against the requirements by simple string matching. This transformation is performed by the *computeResourceCapabilities* and *computeResourceGroupCapabilities* methods of the *WrkCtrSchedulingInteropDataProvider* class, which also must take into account information from the Y table. Consider carefully whether you want the new requirement to be available both for job and operation scheduling. If the requirement must be available for operation scheduling, the used capacity for the group with regards to the Y property must be saved and maintained, along with the capacity reservations, to avoid overbooking. This capability comes at a high performance cost during scheduling.

MorphX model element prefixes for the operations resource framework

All elements that concern the operations resource model are prefixed with WrkCtr*. Most are named very similarly to the conceptual names—except for the Resource entity, which for legacy reasons is stored in the WrkCtrTable table.

For more information about the operations resource model framework, see the following Core Concepts documents on Microsoft Dynamics InformationSource (*http://informationsource.dynamics.com*):

- Allocating resources based on resource requirements
- Operations scheduling based on capabilities

To access these documents, sign in to Microsoft Dynamics InformationSource, click Library, and then type the document title in the Search box.

The dimension framework

The dimension framework provides a method for tracking additional pieces of information like department, cost center, or purpose regarding documents throughout the application. That information can be used in accounting to categorize information.

How the dimension framework works

A *Dimension Attribute* is a type of information that is tracked by the dimension framework. The domain of values for a dimension attribute is defined by the instances of the business entity that exist for the backing business entity type. For instance, the OMOperatingUnit table can provide the list of values for an organization unit dimension. *Dimension Attributes* can be placed in a *Dimension Hierarchy* to indicate ordering. For example, one specialization of a *Dimension Hierarchy* is an *Account Structure*. *Dimension Attributes* can be grouped into a *Dimension Attribute Set*, which is used in some *Setup Data* to specify the *Dimension Attributes* that apply in particular situations; for example, the check box next to each dimension on the LedgerAllocation form.

Figure 19-8 shows the conceptual domain model for the dimension framework.

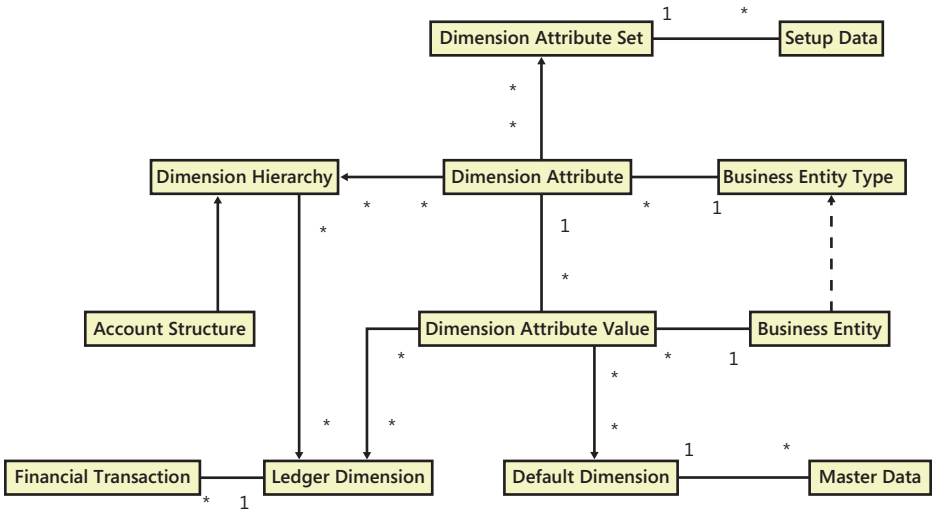


FIGURE 19-8 The dimension framework.

There are four primary storage patterns for exposing and tracking dimension information:

- *Ledger Dimensions* are ordered sets of *Dimension Attribute Values* that are constrained by an account structure and additional accounting rules; for example, *Sales-11005-NorthAmerica-Xbox-70004*. This pattern is normally used on financial data such as journal lines.
- *Default Dimensions* are unordered, unconstrained sets of *Dimension Attribute Values*. For example, a record in the CustTable table may be set to *SalesRegion=NorthAmerica*. This value would then be defaulted into the *Ledger Dimension* when the customer record was used on a sales order. When *Default Dimensions* are shown on a form, all dimensions that are in use by the chart of accounts for the current legal entity are shown.
- *Dimension Attribute Sets* are unordered sets of *Dimension Attributes* that have an enumeration value associated with each *Dimension Attribute*. For example, in the allocation process, the user can mark which *Dimension Attributes* should default from the original transaction and which should take on a specific value. This pattern is used infrequently.
- *Dimension Sets* are ordered sets of dimension values similar to *Ledger Dimensions*, but without the requirement that they contain a main account. They are used primarily for reporting and balance tracking. For example, in order to view the trial balance list page by main account and department, a dimension set containing those two dimensions would be created.

These four primary patterns are further specialized into dozens of specific uses. A few of the more common specializations are as follows:

- The *Default Account* pattern is a specialization of the *Ledger Dimension* storage pattern. Instances of the *Default Account* pattern are stored like a standard ledger account but only

contain a value for the main account *Dimension Attribute*. This pattern is used in areas such as posting profiles to specify which main account is used when a *Ledger Dimension* is created by financial processes.

- The *Dynamic Account* pattern is a specialization of the *Ledger Dimension* storage pattern. This pattern is used on journals where an *Account Type* field is available. When the *Account Type* is set to *Ledger*, then it behaves like a standard *Ledger Dimension* account. When the account type is set to something else, like *Customer*, it acts as a customer lookup. When a non-ledger type is used, a predefined hidden *Dimension Attribute* is used to signify customer, vendor, item, or whatever type is used.

In addition, a variety of budgeting patterns mirror the accounting patterns.

Constrain combinations of values

You can constrain the combinations of values that are valid in *Ledger Dimensions* in two ways.

If constraints are set up in the tree in the Configure Account Structure form (General Ledger > Setup > Financial Dimensions > Configure Account Structures), these constraints are stored in the DimensionConstraintNode and DimensionConstraintNodeCriteria tables. Because the structure of the data in these tables is highly complex, it is much easier to use the *DimensionValidation::validateByTree* method to perform validation rather than reading the constraint node tables directly. The *validateByTree* method validates that a *Ledger Dimension* matches the constraints specified in these tables.

The other method of constraining values is to click the Relationships button on the Action Pane of the Configure Account Structures form, and then use the Select Relationships form to specify the relationships that you want to apply to the account structure. The Select Relationships form shows all of the organization model hierarchies that contain organization model types used as the backing entities for *Dimension Attributes* in the current hierarchy. For example, if an account structure contains departments and cost centers, and an organization model exists that relates departments to cost centers, then that information appears in this form. The information will appear twice with party A and party B reversed. This allows a system administrator to specify whether departments must be parents or children of a given cost center to be valid. These organization model constraints are similarly applied when you use the *DimensionValidation* methods.

Create values

You can create *Ledger Dimensions* programmatically in two ways. To explicitly create them, use the *DimensionStorage* class. You can use this class to add multiple hierarchies and values. When you call the *save* method, it attempts to find an existing combination. If no combination is found, a new one

is created. *Ledger Dimensions* are immutable and only one exists for any given combination. So if the same account is used twice, this method guarantees that only one instance is created in the database.

When working with existing default accounts and ledger dimensions, you can use the *DimensionDefaultingService* class to combine the values into new combinations. For example, the *DimensionDefaultingService*::*serviceCreateLedgerDimension* method takes a default account and one or more Default Dimensions and combines them to form a full *Ledger Dimension*.

Extend the dimension framework

The most common customization of the dimension framework is to add a new backing entity type. Microsoft Dynamics AX 2012 includes approximately 30 backing entities. To add a backing entity type, the only requirement is that the entity must have a natural key that consists of a unique, single-part string that is 30 characters or less.

To add a new backing entity type, create a view to wrap the entity that meets the following criteria:

- The view name must be DimAttribute*[entityname]*. For example, DimAttributeCustTable.
- The view must contain a root data source named BackingEntity, which is backed by the table containing the surrogate key and the natural key.
- The view can optionally contain additional related data sources to handle inheritance or relational associations to provide additional fields, such as a name from the table DirPartyTable.
- The view must contain the following fields named exactly as follows:
  - **Key** Must be the surrogate key field of the backing entity; for example, an *Int64 RecId* field.
  - **Value** Must be the natural key field of the backing entity; for example, a *str30 AccountNum* field.
  - **Name** Must point to the additional description for the entity; for example, a *str60* description field.

If the view meets these criteria, the entity will automatically become available as a backing entity type:

> Because the list of backing entity types are cached both on the client and server, a new type does not appear in the list of existing entities until a call to clear the caches is performed, or until both the client and server are restarted. To clear the caches and have the new entity type appear immediately, use the options on the Tools > Caches menu in the Development Workspace.

Query data

Dimension Attributes are data and can be added or removed by the user. This means that specific dimensions should not be referenced directly in code because there is no guarantee that a given dimension exists. Instead, treat dimension references as configurable data. The one exception to this rule is the *Main Account Dimension Attribute*. All installations are guaranteed to have exactly one *Dimension Attribute* that is backed by *Main Account*. To retrieve this *Dimension Attribute*, use the *DimensionAttribute::getMainAccountDimensionAttribute* method.

Querying dimension information depends on the pattern being used. In the case of a *Ledger Dimension*, either the full combination can be used or the constituent parts. To get the full concatenated combination, create a join to the DimensionAttributeValueCombination table, as shown in the following example:

```
GeneralJournalAccountEntry          gjae;
DimensionAttributeValueCombination  davc;

select gjae join DisplayValue from davc where
    davc.RecId == gjae.LedgerDimension;
```

To get a constituent part of the Ledger Dimension, you can use the DimensionAttributeLevelValueView abstraction to abstract some of the complexity of the dimension model:

```
GeneralJournalAccountEntry          gjae;
DimensionAttributeLevelValueView    dalvv;
DimensionAttribute                  department;

department = DimensionAttribute::findByName('Department');

select gjae join DisplayValue from dalvv where
    dalvv.ValueCombinationRecId == gjae.LedgerDimension &&
    dalvv.DimensionAttribute == department.RecId;
```

The main account *Dimension Attribute* is a special case. This *Dimension Attribute* has been denormalized to the DimensionAttributeValueCombination table to optimize the performance of queries for this value, because it is the most often used:

```
GeneralJournalAccountEntry          gjae;
DimensionAttributeValueCombination  davc;
MainAccount                         mainAccount;

select gjae
    join MainAccount from davc where
        davc.RecId == gjae.LedgerDimension
    join Name from mainAccount where
        mainAccount.RecId == davc.MainAccount;
```

You query *Default Dimensions* in a similar way to *Ledger Dimensions*; however, *Default Dimensions* do not have a concatenated representation because they are unordered sets.

The DimensionAttributeValueSetItemView abstraction joins the DimensionAttributeValueSetItem and DimensionAttributeValue tables to simplify queries:

```
CustTable                           custTable;
DimensionAttributeValueSetItemView  davsiv;
DimensionAttribute                  department;

department = DimensionAttribute::findByName('Department');

select custTable
    join DisplayValue from davsiv where
        davsiv.DimensionAttributeValueSet == custTable.DefaultDimension &&
        davsiv.DimensionAttribute == department.RecId;
```

Physical table references

Table 19-2 maps the concept names in the conceptual domain model to the names of physical table elements that realize these concepts in the application where the names are not the same.

TABLE 19-2 Mapping between concepts and physical tables.

Concept name	Physical tables
Ledger Dimension	DimensionAttributeValueCombination, DimensionAttributeValueGroupCombination, DimensionAttributeValueGroup, DimensionAttributeLevelValue
Default Dimension	DimensionAttributeValueSet, DimensionAttributeValueSetItem
Dimension Attribute Set	DimensionAttributeSet

For more information about the dimension framework, download the following white papers:

- "Securing Data by Dimension Value by Using Extensible Data Security (XDS)" at *http://www.microsoft.com/download/en/details.aspx?id=26921*.
- "Implementing the Account and Financial Dimensions Framework" at *http://technet.microsoft.com/en-us/library/hh272858.aspx*.

The accounting framework

The accounting framework uses policies and rules to derive accounting requirements for amounts and business events that are documented on source document lines. These policies and rules are abstracted as five categories:

- ***Accounting Policy*** Used to determine if accounting applies for a business event – monetary amount combination.

- ***Main Account Derivation Rule*** Used to determine main account values.
- ***Main Account Dimension List Provider*** Used to provide a list of main accounts and side (debit or credit) combinations.
- ***Dimension Derivation Rule*** Used to determine dimension values.
- ***Accounting Journalization Rule*** Used to determine which main account dimension list provider should be used and to determine journalization parameters, such as the posting type, that should be used.

The accounting framework is also responsible for transferring *Subledger Journal* entries to the *General Journal*. Rules for *Subledger Journal* transfers are specified by legal entity and source document type, and they determine when the *Subledger Journal* is transferred to the *General Journal* and whether summarization occurs on transfer.

How the accounting framework works

The *Accounting Distribution* process creates at least one *Accounting Event*. An *Accounting Event* groups a set of distributions based on their accounting date. When a *Source Document* header is submitted to a *Processor* for processing and the *Processor* transitions the document from an *In Process* state to a *Completed* state, the *Journalization Processor* (journalizer) is called. The journalizer processes all *Accounting Events* associated with the document that are in a *Started Process State*, and transitions them to a *Journalized Process State*. An *Accounting Policy* determines whether accounting is required for amounts and business events that are documented on a *Source Document Line*. If the *Accounting Policy* specifies that accounting is required, the journalizer uses *Journalization Rules*, *Main Account Derivation Rules*, *Dimension Derivation Rules*, and the *Main Account Dimension List Provider* to determine the main account–dimension combinations to use when creating balanced subledger journal entries.

Figure 19-9 shows the conceptual domain model for the accounting framework.

Subledger Journal Transfer Rules, shown in Figure 19-10, specify when the transfer should occur (synchronous, asynchronous, or scheduled batch) and whether amounts for the same main account – dimension combination should be summarized when transferred to the general journal.

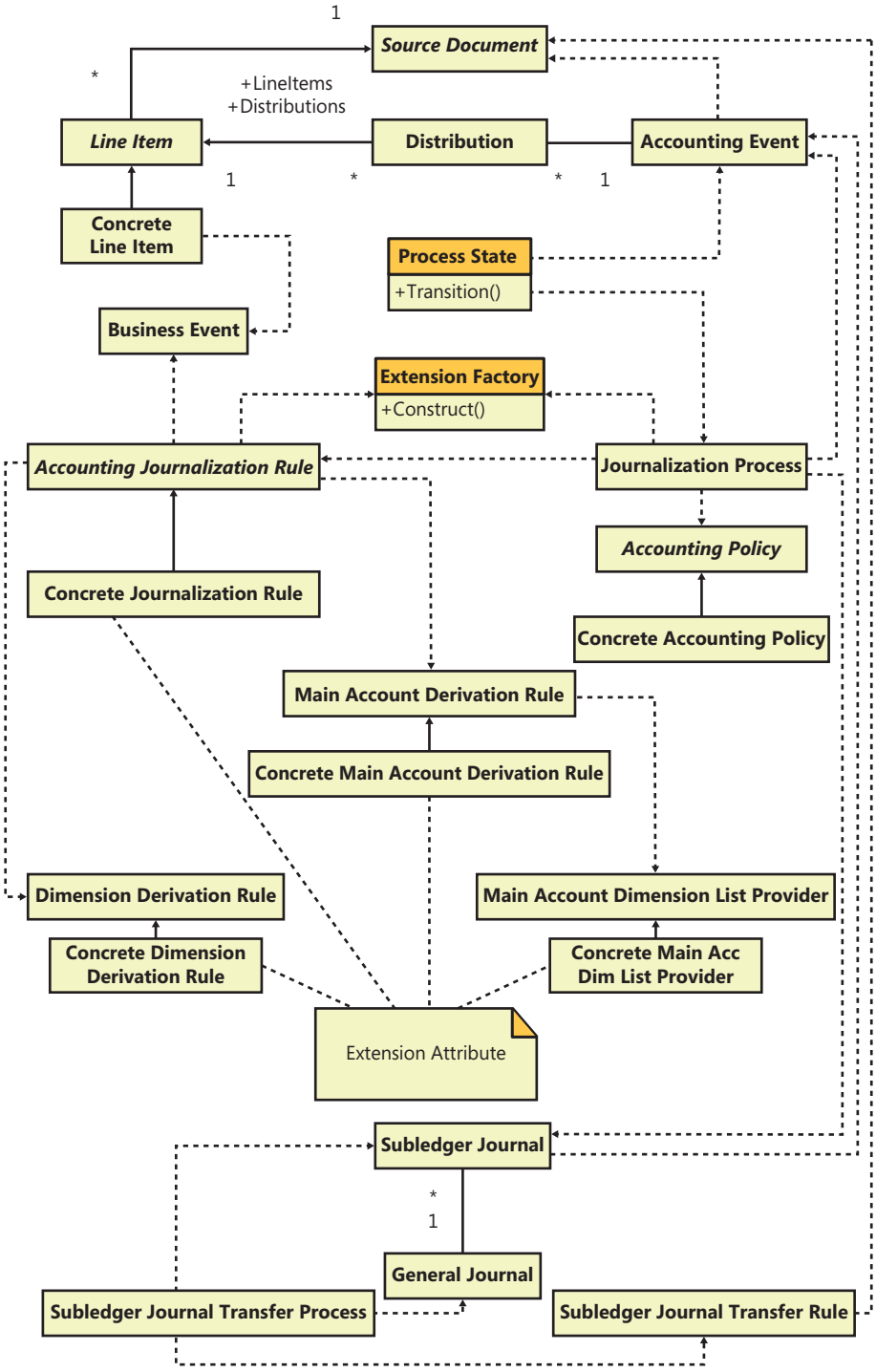


FIGURE 19-9 The accounting framework.

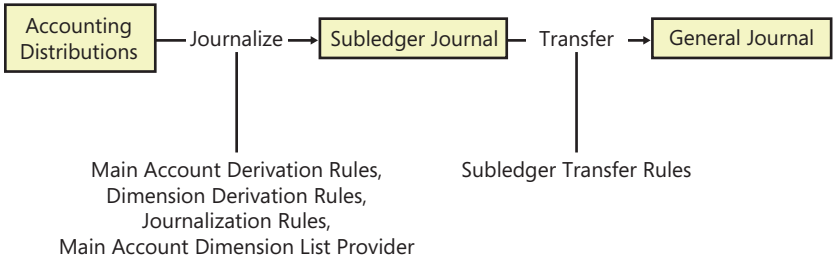


FIGURE 19-10 Rule application in the accounting process.

When to use the accounting framework

Extend the accounting framework to create concrete implementations of accounting policy, journalization, main account derivation, main account dimension list providers, and dimension derivation rules to support new source document implementations. In Microsoft Dynamics AX 2012, the accounting framework has been extended to create concrete accounting policies, journalization, main account derivation, and dimension derivation rules used to generate subledger journal entries on the purchase requisition, purchase order, product receipt, vendor invoice, travel requisition, expense report, and free-text invoice source documents.

Extensions to the accounting framework

The Microsoft Dynamics AX 2012 purchase requisition (PurReqSourceDocument prefix) is an example of an extension of the source document framework components. The AccPolicyCommitFunds-ExpensedProd accounting policy and AccJourRuleCommitFundsForExpProdExtPrice dimension derivation rule are extensions to the accounting framework that specify the accounting requirements for the purchase requisition document. These are examples of extensions to the accounting framework.

Accounting framework process states

The *Process States* for the accounting process are illustrated in Figure 19-11.

Each processing state performs an action and updates the status of the accounting event that is being processed. Table 19-3 describes the process states.

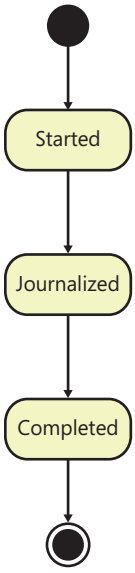


FIGURE 19-11 State model for the accounting process.

TABLE 19-3 Process states for the accounting framework.

State	Process	Description
Started	Accounting distribution process	The accounting event is created and is awaiting journalization.
Journalized	Subledger journalizing process	Subledger journal entries have been created, recording the accounting impact of the distributions that are associated with the accounting event. The subledger journal entries have not been transferred to the general journal.
Completed	Subledger journal transfer process	The subledger journal entries that are associated with the accounting event have been transferred to the general journal.

MorphX model element prefixes for the accounting framework

Table 19-4 maps the concept names in the conceptual domain model to the prefixes added to the names of MorphX model elements that realize these concepts in the application.

TABLE 19-4 Mapping between accounting framework concepts and prefixes of MorphX model elements.

Concept	MorphX model element prefix
Accounting Policy	*AccPolicy*
Subledger Journal Transfer Process	*SubledgerJournalTransfer*
Accounting Event	*AccountingEvent*

Concept	MorphX model element prefix
Accounting Journalization Rule	*AccJourRule*
Dimension Derivation Rule	*DimensionDerivationRule*
Main Account Derivation Rule	*MainAccountDerivationRule*
Main Account Dimension List Provider	*MainAccountDimensionListProvider*
Journalization Process	*SubledgerJournalization* *SubledgerJournalizer*
Subledger Journal	*SubledgerJournal*
General Journal	*GeneralJournal*
Subledger Journal Transfer Rule	*SubledgerJournalTransferRule*

The source document framework

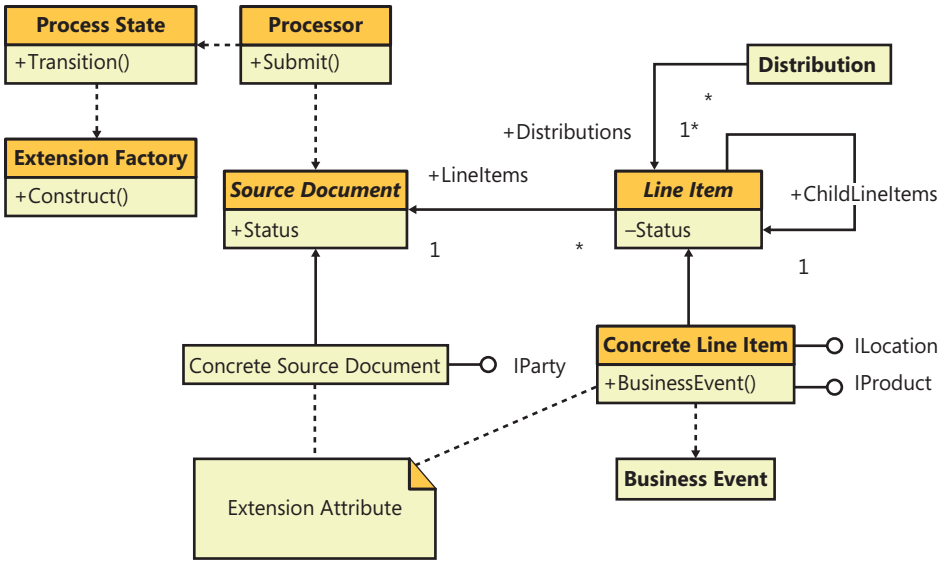
A *Source Document* is an original record that documents the occurrence of one or more *Business Events* in an accounting system. *Concrete Source Documents,* such as purchase orders, product receipts, and vendor invoices, are entered into an accounting system that records, classifies, tracks, and reports on the quantity and value of economic resources that are exchanged or committed for exchange when performing activities identified by *Business Events* such as purchase, product receipt, and payment request.

How the source document framework works

The source document framework generates a projection of a concrete source document for a process that transitions the source document status to reflect the state of the process. Figure 19-12 shows the domain model for the source document framework.

Microsoft Dynamics AX submits a *Source Document* header or line record to a *Processor* for processing when a user confirms that the documentation requirements of business events and internal process controls have been met. A *Processor* is a state machine that transitions the processing of the source document and its lines from one *Process State* to another. The *Processor* creates a *Process State* object that corresponds to the status of the source document or the status of the source document line item and then commands the *Process State* to transition the process to the next state.

A *Process State* first constructs a *Concrete Source Document* or a *Concrete Line Item* from the provided *Source Document* header or line record by using an extension factory facility. The extension factory facility uses the source document type and the table number of the *Concrete Source Document* or the *Concrete Line Item* provided by the header or line record to find a matching concrete source document class. A matching source document class is one that is annotated with a class attribute recognized as an *Extension Attribute* by the *Extension Factory* and that also specifies a matching source document type and table number as arguments.



Concept	Model Element Prefix
Source Document	SourceDocument
Line Item	SourceDocumentLineItem
Processor	SourceDocumentProcessor
Extension Factory	SysExtension
Process State	SourceDocumentState
	SourceDocumentLineState
Distribution	AccountingDistribution
Business Event	BusinessEvent

FIGURE 19-12 The source document domain model.

A *Process State* accesses a data projection of the *Concrete Source Document* and *Concrete Source Document Line*, performs an action, transitions the process to a new state, and updates the status of the process's *Concrete Source Document* or *Concrete Source Document Line* accordingly. The data projections of the *Concrete Source Document* and *Concrete Source Document Line* are defined by one or more interfaces. For example, implementing the *IParty* interface provides a party account number, and implementing the *IProduct* interface provides an item number and a production category to an accessing *Process State*.

When to use the source document framework

Extend the source document framework to implement concrete source documents that document business events whose financial consequences are recorded in the subledger journal. In Microsoft Dynamics AX 2012, the source document framework has been extended to implement the source documents purchase requisition, purchase order, product receipt, vendor invoice, travel requisition, expense report, and free-text invoice.

Extensions to the source document framework

The Microsoft Dynamics AX 2012 free-text invoice (CustInvoiceSourceDocument prefix) is the simplest extension of the source document framework components. Readers new to the source document framework should review this extension of the source document framework first. The concrete source document and concrete line item implement only those source document projection interfaces that are required by the accounting distribution processor and the subledger journalizing processor.

The process states for the subledger journalizing process and the accounting distribution process are illustrated in Figure 19-13. Each processing state performs an action and updates the status of the source document or source document line item that participated in the process. Table 19-5 describes the states of the subledger journalizing process and the accounting distribution process.

TABLE 19-5 States of the subledger journalizing and accounting distribution processes.

State	Process	Description
In Process	Subledger journalizing process	The state reached when a source document is first created and when it is changed. The subledger journalizing process transitions to the completed state when the original source document or a changed source document is confirmed.
Completed	Subledger journalizing process	The state reached when a source document is confirmed and the documented consequences of business events are journalized. The subledger journalizing process transitions to the In Process state when a source document is changed or when a source document is finalized and can no longer be changed.
Finalized	Subledger journalizing process	The state reached when a source document can no longer be changed. The subledger journalizing process balances any open account entries when finalizing the source document.
Draft	Accounting distribution process	The state reached when a source document line is first created or when all accounting distributions that reference a source document line are deleted.
Fully Distributed	Accounting distribution process	The state reached when an accounting distribution is first added to distribute an amount that is documented on a source document line; for example, a discount. This state is also reached when an accounting distribution is generated or derived from amounts documented on a source document line; for example, an extended price. This state is also reached when the sum of the distribution amounts equals the distributed amount, or when a source document is changed.
Partly Distributed	Accounting distribution process	The state reached when the sum of the distribution amounts does not equal the distributed amount.
Cancelled	Subledger journalizing process Accounting distribution process	The state reached when a source document or source document line item is cancelled. All accounting distributions are reversed and the consequences of the cancelled business event are journalized before this state is reached.

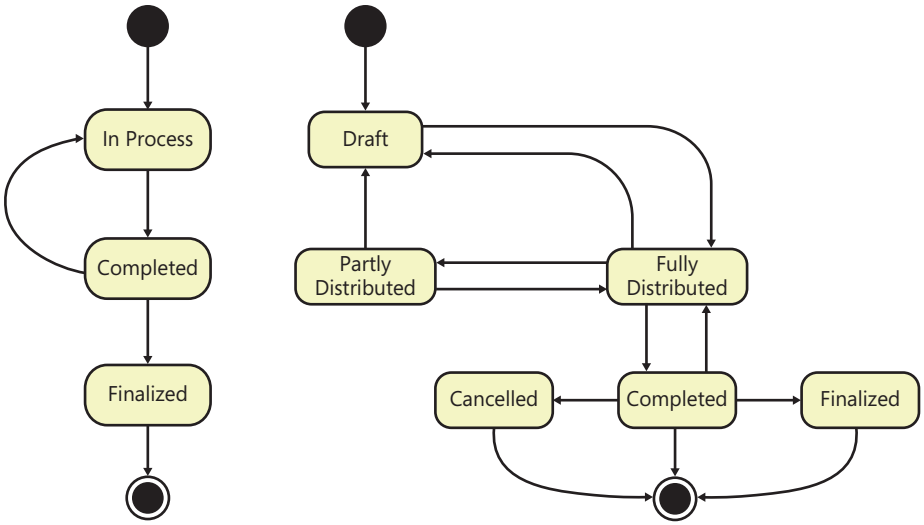


FIGURE 19-13 State model for the documentation process, the accounting distribution process, and the subledger journalizing process.

MorphX model element prefixes for the source document framework

Table 19-6 maps the concept names in the conceptual domain model for the source document framework to the prefixes added to the names of MorphX model elements that realize these concepts in the application.

TABLE 19-6 Mapping between source document framework concepts and prefixes for MorphX model elements.

Concept	MorphX model element prefix
Source Document	*SourceDocument*
Line Item	*SourceDocumentLineItem*
Processor	*SourceDocumentProcessor*
Process State	*SourceDocumentState* *SourceDocumentLineState*
Business Event	*BusinessEvent*
Distribution	*AccountingDistribution*
Extension Factory	*SysExtension*

CHAPTER 20

Reflection

In this chapter

Introduction 669
Reflection system functions 670
Reflection APIs 673

Introduction

Reflection is a programmatic discoverability mechanism of a type system. By using the reflection application programming interfaces (APIs) of the Microsoft Dynamics AX application model, you can read and traverse element definitions as though they were in a table, an object model, or a tree structure.

You can perform interesting analyses with the information that you get through reflection. The Reverse Engineering tool provides an excellent example of the power of reflection. Based on element definitions in MorphX, the tool generates Unified Modeling Language (UML) models and entity relationship diagrams (ERDs) that you can browse in Microsoft Visio.

You can also use reflection to invoke methods on objects. This capability is of little value to business application developers. But for framework developers, the power to invoke methods on objects can be valuable. Suppose you want to programmatically write any record to an XML file that includes all of the fields and display methods. With reflection, you can determine the fields and their values and also invoke the display methods to capture their return values.

X++ features a set of system functions that you can use for reflection, in addition to three reflection APIs. The reflection system functions are as follows:

- **Intrinsic functions** A set of functions that you can use to refer to an element's name or ID safely
- ***typeOf* system function** A function that returns the primitive type for a variable
- ***classIdGet* system function** A function that returns the ID of the class for an instance of an object

The reflection APIs are as follows:

- **Table data** A set of tables that contains all element definitions. The tables provide direct access to the contents of the model store files. You can query for the existence of elements

and certain properties, such as model, *created by*, and *created datetime*. However, you can't retrieve information about the contents or structure of each element.

- **Dictionary** A set of classes that provide a type-safe mechanism for reading metadata from an object model. Dictionary classes provide basic and more abstract information about elements in a type-safe manner. With few exceptions, this API is read-only.
- **Treenodes** A class hierarchy that provides the Application Object Tree (AOT) with an API that can be used to create, read, update, and delete any piece of metadata or source code. This API can provide all information about anything in the AOT. You navigate the treenodes in the AOT through the API and query for metadata in a non-type-safe manner.

This chapter delves into the details of these system functions and APIs.

Reflection system functions

The X++ language features a set of system functions that can be used to reflect on elements. They are described in the following sections.

Intrinsic functions

Use intrinsic functions whenever you need to reference an element from within X++ code. Intrinsic functions provide a way to make a type-safe reference. The compiler recognizes the reference and verifies that the element being referenced exists. If the element doesn't exist, the code doesn't compile. Because elements have their own lifecycles, a reference doesn't remain valid forever; an element can be renamed or deleted. Using intrinsic functions ensures that you are notified of any broken references at compile time. A compiler error early in the development cycle is always better than a run-time error later.

All references you make using intrinsic functions are captured by the Cross-reference tool. You can determine where any element is referenced, regardless of whether the reference is in metadata or code. The Cross-reference tool is described in Chapter 2, "The MorphX development environment and tools."

Consider these two implementations:

```
print "MyClass";          //Prints MyClass
print classStr(MyClass);  //Prints MyClass
```

Both lines of code have the same result: the string *MyClass* is printed. As a reference, the first implementation is weak. It will eventually break if the class is renamed or deleted, meaning that you'll need to spend time debugging. The second implementation is strong and unlikely to break. If you were to rename or delete *MyClass*, you could use the Cross-reference tool to analyze the impact of your changes and correct any broken references.

By using the intrinsic functions *<Concept>Str*, you can reference all elements in the AOT by their names. You can also use the intrinsic function *<Concept>Num* to reference elements that have an ID. Intrinsic functions are not limited to root elements; they also exist for class methods, table fields, indexes, and methods. More than 50 intrinsic functions are available. Here are a few examples:

```
print fieldNum(MyTable, MyField);   //Prints 60001
print fieldStr(MyTable, MyField);   //Prints MyField
print methodStr(MyClass, MyMethod); //Prints MyMethod
print formStr(MyForm);              //Prints MyForm
```

The ID of an element is assigned when the element is created in the model store. In the preceding example, the ID *60001* is assigned to the first element field created in a table. (Element IDs are explained in Chapter 21, "Application models.")

Two other intrinsic functions are worth noting: *identifierStr* and *literalStr*. The *identifierStr* function allows you to refer to elements if a more feature-rich intrinsic function isn't available. The *identifierStr* function provides no compile-time checking and no cross-reference information. However, using the *identifierStr* function is still better than using a literal because the intention of referring to an element is captured. If a literal is used, the intention is lost—the reference might be to user interface text, a file name, or something completely different. The Best Practices tool detects the use of *identifierStr* and issues a best practice warning.

The Microsoft Dynamics AX runtime automatically converts any reference to a label ID to its corresponding label text. In most cases, this behavior is what you want; however, you can prevent the conversion by using *literalStr*. The *literalStr* function allows you to refer to a label ID without converting the label ID to the label text, as shown in this example:

```
print "@SYS1";             //Prints Time transactions
print literalStr("@SYS1"); //Prints @SYS1
```

In the first line of the example, the label ID (*@SYS1*) is automatically converted to the label text (*Time transactions*). In the second line, the reference to the label ID isn't converted.

typeOf system function

The *typeOf* system function takes a variable instance as a parameter and returns the base type of the parameter. Here is an example:

```
int i = 123;
str s = "Hello world";
MyClass c;
guid g = newGuid();

print typeOf(i);  //Prints Integer
```

```
print typeOf(s);  //Prints String
print typeOf(c);  //Prints Class
print typeOf(g);  //Prints Guid
pause;
```

The return value is an instance of the *Types* system enumeration. It contains an enumeration for each base type in X++.

classIdGet system function

The *classIdGet* system function takes an object as a parameter and returns the class ID for the class element of which the object is an instance. If the parameter passed is null, the function returns the class ID for the declared type, as shown in this example:

```
MyBaseClass c;
print classIdGet(c);  //Prints the ID of MyBaseClass

c = new MyDerivedClass();
print classIdGet(c);  //Prints the ID of MyDerivedClass
pause;
```

This function is particularly useful for determining the type of an object instance. Suppose you need to determine whether a class instance is of a particular class. The following example shows how you can use *classIdGet* to determine the class ID of the *_anyClass* variable instance. If the *_anyClass* variable really is an instance of *MyClass*, it's safe to assign it to the *myClass* variable.

```
void myMethod(object _anyClass)
{
    MyClass myClass;
    if (classIdGet(_anyClass) == classNum(MyClass))
    {
        myClass = _anyClass;
        ...
    }
}
```

Notice the use of the *classNum* intrinsic function, which evaluates the parameter at compile time, and the use of *classIdGet*, which evaluates the parameter at run time.

Because inheritance isn't taken into account, this sort of implementation is likely to break the object model. In most cases, any instance of a derived *MyClass* class should be treated as an actual

MyClass instance. The simplest way to handle inheritance is to use the *is* and *as* operators. For more information, see Chapter 4, "The X++ programming language."

Note This book promotes customization through inheritance by using the Liskov substitution principle.

Reflection APIs

The X++ system library includes three APIs that can be used to reflect on elements. They are described in the following sections.

Table data API

Suppose that you want to find all classes whose names begin with *Invent*. The following example shows one way to conduct your search:

```
static void findInventoryClasses(Args _args)
{
    SysModelElement modelElement;

    while select name from modelElement
        where modelElement.ElementType == UtilElementType::Class
           && modelElement.Name like 'Invent*'
    {
        info(modelElement.Name);
    }
}
```

The SysModelElement table provides access to all elements. The *ElementType* field holds the concept to search for. The data model for the model store contains nine tables, which are shown and explained in Figure 20-1.

Note The UtilElements table is still available for backward compatibility. It is implemented as an aggregated view on top of the SysModel tables. For performance reasons, you should limit usage of this compatibility feature and eventually rewrite your code to use the new API.

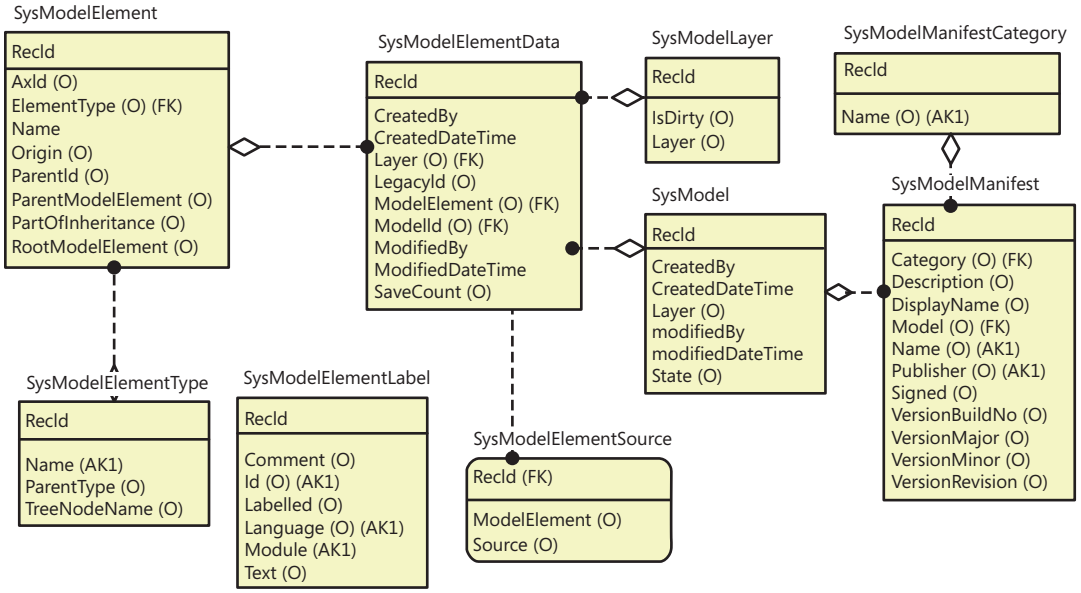


FIGURE 20-1 The data model for the model store.

Because of the nature of the table data API, the SysModel tables can also be used as data sources in a form or a report. A form showing the table data is available from Tools > Model Management > Model elements. In the form, you can use standard query capabilities to filter and search the data.

The SysModelElement table contains all of the elements in the model store; it is related to the SysModelElementData table, which contains the various definitions of each element. For each SysModelElement record, there is at least 1 SysModelElementData record—and perhaps as many as 16 if the element is customized across all 16 layers. In other words, the element defines the customization granularity. You cannot customize less than an element. For example, even if you change just one property on an element, a new record is inserted into the SysModelElementData table that includes all properties of the element.

Note System elements, as listed under *System Documentation* node in the AOT, are not present in these tables.

Elements are structured in hierarchies. The root of a hierarchy is the root element; for example, a form. The form contains data source, control, and method elements. The hierarchy can encompass multiple levels; for example, a form control can have methods. The root element and parent element are exposed in the *RootModelElement* and *ParentModelElement* fields of the SysModelElement table. The job on the next page finds all elements under the *CustTable* form element and lists the name and type of each element, the name of the parent element, and the AOT path of the associated *TreeNode* class.

```
static void findElementsOnCustTable(Args _args)
{
    SysModelElement modelElement;
    SysModelElement rootModelElement;
    SysModelElement parentModelElement;
    SysModelElementType modelElementType;

    while select name from modelElement
        join Name from modelElementType
            where modelElementType.RecId == modelElement.ElementType
        join name from parentModelElement
            where parentModelElement.RecId == modelElement.ParentModelElement
        exists join rootModelElement
            where rootModelElement.RecId == modelElement.RootModelElement
               && rootModelElement.Name == formStr(CustTable)
               && rootModelElement.ElementType == UtilElementType::Form
    {
        info(strFmt("%1, %2, %3, %4",
            parentModelElement.Name, modelElementType.Name, modelElement.Name,
            SysTreeNode::modelElement2Path(modelElement)));
    }
}
```

Notice the use of the *ElementType* field in the two preceding examples. If the element type is a *UtilElement*, you will find a matching entry in the *UtilElementType* enum; alternatively, you can always join to the SysModelElementType table, which contains information about all element types. All root elements and a few former subelements are *Utilelements*. You can access them through the legacy UtilElements table. Data models with higher fidelity were introduced in this release to support more granular customizations, which among other things facilitate easier upgrade and simpler side-by-side installation of models. For more information, see Chapter 21.

Table 20-1 lists the reflection tables and views. See Figure 20-1 to learn how these tables relate to each other.

TABLE 20-1 Reflection tables and views.

Table or View Name	Description
SysModel	Table containing the models in the model store.
SysModelElement	Table containing the elements in the model store. There is exactly one record for each element in the AOT, regardless of customizations.
SysModelElementData	Table containing the element definitions in the model store. There is one record for each element in each layer.
SysModelElementLabel	Table containing all label text and comments.
SysModelElementSource	Table containing all X++ source code.
SysModelElementType	Table containing definitions of element types. The information in this table is static and is populated at installation time.

Table or View Name	Description
SysModelLayer	Table containing the layers. The information in this table is static and is populated at installation time. There is a record for each of the 16 layers.
SysModelManifest	Table containing the manifest for the models, such as name, publisher, and version number. There is one record for each model.
SysModelManifestCategory	Table containing the categories that a model can belong to. Each model belongs to a category: Standard, Hotfix, Virtual, or Temporary.
UtilElements, UtilIdElements	Aggregated views on top of the SysModel tables. These views are provided for backward compatibility.
UtilModels	View on top of SysModel, SysModelManifest, and SysModelLayer, which make querying models easier.

Note Alternative versions of the tables in Table 20-1 exist. If you postfix the table name with the word *old*, you can access the baseline model store instead of the primary model store. For example, the SysModelElementOld table contains the model elements in the baseline model store. The baseline model store is primarily used in upgrade scenarios.

You can use the *Microsoft.Dynamics.AX.Framework.Tools.ModelManagement* namespace provided by the *AxUtilLib.dll* assembly to create, import, export, and delete models. This assembly can be used from X++—the *SysModelStore* class wraps some of the functionality for easier consumption in X++.

Note When you use the table data API in an environment with version control enabled, the values of some of the fields are reset during the build process. For file-based version control systems, the build process imports .xpo files into empty layers in Microsoft Dynamics AX. The values of the *CreatedBy*, *CreatedDateTime*, *ModifiedBy*, and *ModifiedDateTime* fields are set during this import process and therefore don't survive from build to build.

Dictionary API

The Dictionary API is a type-safe reflection API that can reflect on many elements. The following code example is a revision of the preceding example that finds inventory classes by using the dictionary API. This API gives you access to more detailed type information. This example lists only abstract classes that start with the string *Invent*:

```
static void findAbstractInventoryClasses(Args _args)
{
    Dictionary dictionary = new Dictionary();
    int i;
    DictClass dictClass;
```

```
    for(i=1; i<=dictionary.classCnt(); i++)
    {
        dictClass = new DictClass(dictionary.classCnt2Id(i));

        if (dictClass.isAbstract() &&
            strStartsWith(dictClass.name(), 'Invent'))
        {
            info(dictClass.name());
        }
    }
}
```

The *Dictionary* class provides information about which elements exist and even includes system elements. For example, with this information, you can instantiate a *DictClass* object that provides information about the class, such as whether the class is abstract, final, or an interface; which class it extends; whether it implements any interfaces; what attributes it is decorated with; and what methods it includes. Notice that the *DictClass* class can also reflect on interfaces. Also notice how the class counter is converted into a class ID. This conversion is required because the IDs aren't listed consecutively.

When you run this job, you'll notice that it's much slower than the implementation that uses the table data API—at least the first time you run it. The job performs better after the information is cached.

Figure 20-2 shows the object model for the dictionary API. As you can see, some elements can't be reflected upon by using this API.

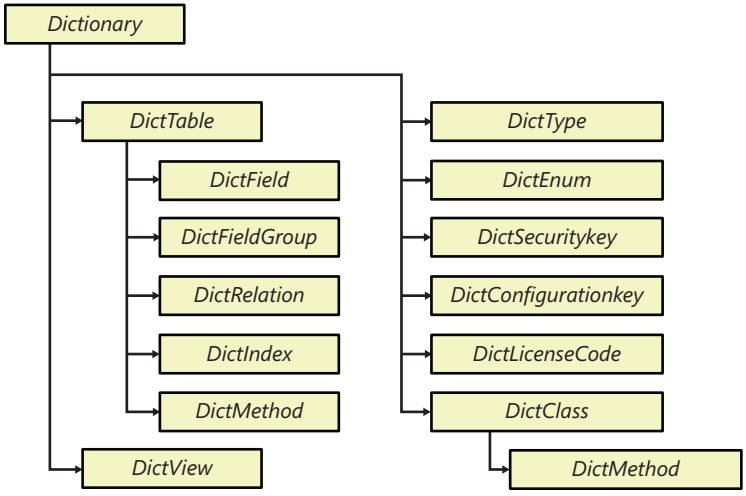


FIGURE 20-2 The object model for the dictionary reflection API.

The following example lists the static methods on the *CustTable* table and reports their parameters:

```
static void findStaticMethodsOnCustTable(Args _args)
{
    DictTable dictTable = new DictTable(tableNum(CustTable));
    DictMethod dictMethod;
    int i;
    int j;
    str parameters;

    for (i=1; i<=dictTable.staticMethodCnt(); i++)
    {
        dictMethod = new DictMethod(
            UtilElementType::TableStaticMethod,
            dictTable.id(),
            dictTable.staticMethod(i));

        parameters = '';
        for (j=1; j<=dictMethod.parameterCnt(); j++)
        {
            parameters += strFmt("%1 %2",
                extendedTypeId2name(dictMethod.parameterId(j)),
                dictMethod.parameterName(j));

            if (j<dictMethod.parameterCnt())
            {
                parameters += ', ';
            }
        }
        info(strFmt("%1(%2)", dictMethod.name(), parameters));
    }
}
```

As mentioned earlier, reflection can also be used to invoke methods on objects. This example invokes the static *find* method on the table *CustTable*:

```
static void invokeFindOnCustTable(Args _args)
{
    DictTable dictTable = new DictTable(tableNum(CustTable));
    CustTable customer;

    customer = dictTable.callStatic(
        tableStaticMethodStr(CustTable, Ffind), '1201');

    print customer.currencyName();    //Prints US Dollar
    pause;
}
```

Notice the use of the intrinsic function *tableStaticMethodStr* to reference the *find* method.

You can also use this API to instantiate class and table objects. Suppose you want to select all records in a table with a given table name. The following example shows you how:

```
static void findRecords(TableId _tableId)
{
    DictTable dictTable = new DictTable(_tableId);
    Common common = dictTable.makeRecord();
    FieldId primaryKeyField = dictTable.primaryKeyField();

    while select common
    {
        info(strFmt("%1", common.(primaryKeyField)));
    }
}
```

First, notice the call to the *makeRecord* method, which instantiates a table cursor object that points to the correct table. You can use the *select* statement to select records from the table. If you want to, you can also insert records by using the table cursor. Notice the syntax used to get a field value out of the cursor object; this syntax allows any field to be accessed by its field ID. This example prints the content of the primary key field. Alternatively, you can use the *getFieldValue* method to get a value based on the name of the field. You can use the *makeObject* method on the *DictClass* class to create an object instance of a class.

All of the classes in the dictionary API discussed so far are defined as system APIs. On top of each of these is an application-defined class that provides even more reflection capabilities. These classes are named *SysDict*<Concept>, and each class extends its counterpart in the system API. For example, *SysDictClass* extends *DictClass*.

Consider the following example. Table fields have a property that specifies whether the field is mandatory. The *DictField* class returns the value of a mandatory property as a bit that is set in the return value of its *flag* method. Testing to determine whether a bit is set is somewhat cumbersome, and if the implementation of the flag changes, the consuming application breaks. The *SysDictField* class encapsulates the bit-testing logic in a *mandatory* method. The following example shows how to use the method:

```
static void mandatoryFieldsOnCustTable(Args _args)
{
    SysDictTable sysDictTable = SysDictTable::newName(tableStr(CustTable));
    SysDictField sysDictField;
    Enumerator enum = sysDictTable.fields().getEnumerator();

    while (enum.moveNext())
    {
        sysDictField = enum.current();
```

```
            if (sysDictField.mandatory())
            {
                info(sysDictField.name());
            }
        }
    }
```

You might also want to browse the *SysDict* classes for static methods. Many of these methods provide additional reflection information and better interfaces. For example, the *SysDictionary* class provides a *classes* method that returns a collection of *SysDictClass* instances. You could use this method to simplify the earlier *findAbstractInventoryClasses* example.

Treenodes API

The two reflection APIs discussed so far have limitations. The table data API can reflect only on the existence of elements and on a small subset of element metadata. The dictionary API can reflect in a type-safe manner, but only on the element types that are exposed through this API.

The treenodes API can reflect on everything, but as always, power comes at a cost. The treenodes API is harder to use than the other reflection APIs, it can cause memory and performance problems, and it isn't type-safe.

In the following code, the example from the "Table data API" section has been revised to use the treenodes API to find inventory classes:

```
static void findInventoryClasses(Args _args)
{
    TreeNode classesNode = TreeNode::findNode(@'\Classes');
    TreeNodeIterator iterator = classesNode.AOTiterator();
    TreeNode classNode = iterator.next();
    ClassName className;

    while (classNode)
    {
        className = classNode.treeNodeName();
        if (strStartsWith(className, 'Invent'))
        {
            info(className);
        }

        classNode = iterator.next();
    }
}
```

First, notice how you find a node in the AOT based on the path as a literal. The *AOT* macro contains definitions for the primary AOT paths. For readability, the examples in this chapter don't use the macro. Also notice the use of a *TreeNodeIterator* class to iterate through the classes.

The following small job prints the source code for the *find* method on the CustTable table by calling the *AOTgetSource* method on the *treenode* object for the *find* method:

```
static void printSourceCode(Args _args)
{
    TreeNode treeNode =
        TreeNode::findNode(@'\Data Dictionary\Tables\CustTable\Methods\find');

    info(treeNode.AOTgetSource());
}
```

The treenodes API provides access to the source code of nodes in the AOT. You can use the class *ScannerClass* to turn the string that contains the source code into a sequence of tokens that can be compiled.

In the following code, the preceding example has been revised to find mandatory fields on the table *CustTable*:

```
static void mandatoryFieldsOnCustTable(Args _args)
{
    TreeNode fieldsNode = TreeNode::findNode(
        @'\Data Dictionary\Tables\CustTable\Fields');

    TreeNode field = fieldsNode.AOTfirstChild();

    while (field)
    {
        if (field.AOTgetProperty('Mandatory') == 'Yes')
        {
            info(field.treeNodeName());
        }

        field = field.AOTnextSibling();
    }
}
```

Notice the alternate way of traversing subnodes. Both this and the iterator approach work equally well. The only way to determine whether a field is mandatory with this API is to know that your node models a field. Field nodes have a property named *Mandatory*, which is set to *Yes* (not to *True*) for mandatory fields.

Use the *Properties* macro when referring to property names. This macro contains text definitions for all property names. By using this macro, you avoid using literal names, like the reference to the *Mandatory* property in the preceding example.

Unlike the dictionary API, which can't reflect all elements, the treenodes API reflects everything. The *SysDictMenu* class exploits this capability, providing a type-safe way to reflect on menus and

menu items by wrapping information provided by the treenodes API in a type-safe API. The following job prints the structure of the MainMenu menu, which typically is shown in the navigation pane:

```
static void printMainMenu(Args _args)
{
    void reportLevel(SysDictMenu _sysDictMenu)
    {
        SysMenuEnumerator enumerator;

        if (_sysDictMenu.isMenuReference() ||
            _sysDictMenu.isMenu())
        {
            setPrefix(_sysDictMenu.label());
            enumerator = _sysDictMenu.getEnumerator();
            while (enumerator.moveNext())
            {
                reportLevel(enumerator.current());
            }
        }
        else
        {
            info(_sysDictMenu.label());
        }
    }

    reportLevel(SysDictMenu::newMainMenu());
}
```

Notice how the *setPrefix* function is used to capture the hierarchy and how the *reportLevel* function is called recursively.

You can also use the treenode API to reflect on forms and reports, and their structure, properties, and methods. The Compare tool in MorphX uses this API to compare any node with any other node. The *SysTreeNode* class contains a *TreeNode* class and implements a cascade of interfaces, which makes *TreeNode* classes consumable for the Compare tool and the Version Control tool. The *SysTreeNode* class also contains a powerful set of static methods.

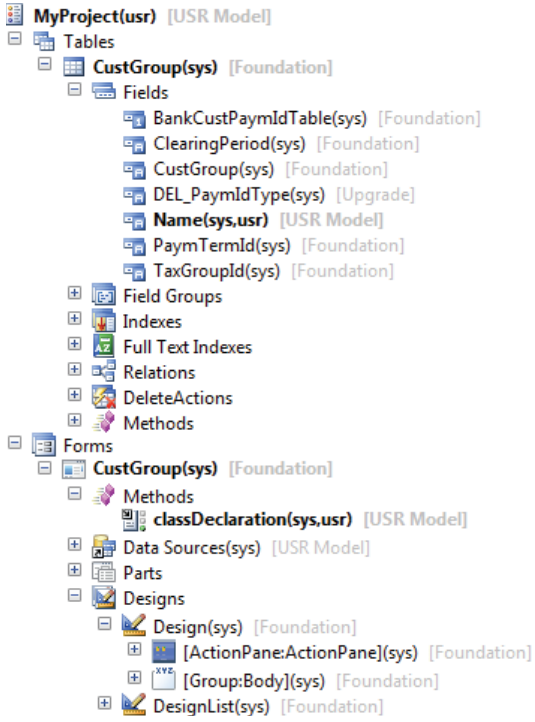
The *TreeNode* class is actually the base class of a larger hierarchy. You can cast instances to specialized *TreeNode* classes that provide more specific functionality. The hierarchy isn't fully consistent for all nodes. You can browse the hierarchy in the AOT by clicking System Documentation, clicking Classes, right-clicking TreeNode, pointing to Add-Ins, and then clicking Type hierarchy browser.

Although this section has only covered the reflection functionality of the treenodes API, you can use the API just as you do the AOT designer. You can create new elements and modify properties and source code. The Wizard Wizard uses the treenodes API to generate the project, form, and class implementing the wizard functionality. You can also compile and get layered nodes and nodes from the baseline model store. The capabilities that go beyond reflection are very powerful, but proceed with great care. Obtaining information in a non-type-safe manner requires caution, but writing in a non-type-safe manner can lead to catastrophic situations.

TreeNodeType

Different types of treenodes have different capabilities. The *TreeNodeType* class can be used to reflect on the treenode. The *TreeNodeType* class provides reliable alternatives to making assumptions about a treenode's capabilities based on its properties. In previous versions of Microsoft Dynamics AX, fragile assumptions could be found throughout the code base; for example, it was assumed that a treenode supported version control if the treenode had a *utilElementType* and no parent ID.

The *TreeNodeType* class provides a method that returns the type identification, plus seven methods that return Boolean values providing information about the treenode's capabilities. The usage of these methods are described later in this section. Figure 20-3 shows the information that the *TreeNodeType* class provides for each treenode in a project containing a table and a form. The left side of the illustration shows a screenshot of the project itself. The right side contains a table that, for each treenode, lists the treenode type ID and capabilities.



ID	isConsumingMemory	isGetNodeInLayerSupported	isLayerAware	isModelElement	isRootElement	isUtilElement	isVCSControllableElement
30							
27							
204	X	X	X	X	X	X	X
110							
416		X	X	X		X	
405		X	X	X		X	
405		X	X	X		X	
405		X	X	X		X	
405		X	X	X		X	
405		X	X	X		X	
405		X	X	X		X	
121							
117							
316							
120							
167							
191							
27							
504	X	X	X	X	X	X	X
191							
201			X	X			
140			X	X			
144							
1435							
141							
148			X	X			
152			X	X			
152			X	X			
148			X	X			

FIGURE 20-3 Information provided by the *TreeNodeType* class for the treenodes in a table and a form.

- ***ID*** The ID of the treenode type is defined in the system and is available in the *TreeNodeSysNodeType* macro. Nodes with the same ID have the same behavior.
- ***isConsumingMemory*** Tree nodes in MorphX contain data that the Microsoft Dynamics AX runtime doesn't manage, and the memory for a node isn't automatically deallocated. For each node where *isConsumingMemory* is true, you should call the *treenodeRelease* method to free the memory when you no longer reference any subnodes. Alternatively, you can use the *TreeNodeTraverser* class because the class will handle this task for you. For an example of this, see the *traverseTreeNodes* method of the *SysBpCheck* class.
- ***isGetNodeInLayerSupported*** With treenodes that support the *getNodeInLayer* method, you can navigate to versions of the node in other layers. In other words, you can access the nodes in the lower layers by using this method.
- ***isLayerAware*** Treenodes that are layer-aware display a layer indicator in the AOT; for example, *SYS* or *USR*. You can retrieve the layer of a node by using the *AOTLayer* method, and you can retrieve all layers that are available by using the *AOTLayers* method. Note that the *AOTLayers* method does not roll up layers for subnodes: this method returns what is shown in the AOT. The roll-up layer information is available through the *ApplObjectLayerMask* method, which is used in the AOT to determine whether a node is shown in bold. If a node is bold in the AOT, either the node itself or one of its subnodes is present in the current layer.
- ***isModelElement*** Treenodes that are model elements are represented by a record in the SysModelElement table.
- ***isRootElement*** A root element is placed in the root of the treenode hierarchy, and the *RootModelElement* field for all submodel elements references the root element's *recid*.
- ***isUtilElement*** If the treenode is a *UtilElement*, a corresponding record can be found in the *UtilElements* view. Further, the primary key information can be retrieved through the treenode's *utilElement* method.
- ***isVCSControllableElement*** You can use the *isVCSControllableElement* method to determine the granularity of the file-based artifacts that are stored in a version control system. In most cases the granularity under version control is per root element; in other words, you are working on entire forms, classes, and tables under version control. However, for Microsoft Visual Studio elements, the granularity is different, and you are able to work on individual Visual Studio files; for example, .cs files.

```
if (treenode.treeNodeType().isVCSControllableElement())
{
    versionControl.checkOut(treenode);
}
```

The following example shows how to access the type information for a treenode:

```
static void GetTreeNodeTypeInfo(Args _args)
{
    TreeNode treeNode = TreeNode::findNode(
        @'\Data Dictionary\Tables\CustTable\Methods\find');
    TreeNodeType treeNodeType = treeNode.treeNodeType();

    info(strFmt("Id: %1", treeNodeType.id()));
    info(strFmt("IsConsumingMemory: %1", treeNodeType.isConsumingMemory()));
    info(strFmt("IsGetNodeInLayerSupported: %1",
        treeNodeType.isGetNodeInLayerSupported()));
    info(strFmt("IsLayerAware: %1", treeNodeType.isLayerAware()));
    info(strFmt("IsModelElement: %1", treeNodeType.isModelElement()));
    info(strFmt("IsRootElement: %1", treeNodeType.isRootElement()));
    info(strFmt("IsUtilElement: %1", treeNodeType.isUtilElement()));
    info(strFmt("IsVCSControllableElement: %1",
        treeNodeType.isVCSControllableElement()));
}
```

Note You can use the *TreeNodeType* class to reflect on the meta model. This class functions on a higher level of abstraction—instead of reflecting on the elements in the AOT, it reflects on element types. The *SysModelMetaData* class provides another way of reflecting on the meta model.

CHAPTER 21

Application models

In this chapter

Introduction 687
Layers 688
Models 690
Element IDs 692
Create a model 693
Prepare a model for publication 694
Upgrade a model 699
Move a model from test to production 700
Model store API 703

Introduction

Microsoft Dynamics AX 2012 introduces a new era for managing metadata artifacts.

In previous versions of the product, metadata artifacts were stored in Application Object Data (AOD) files. These files served two purposes. First, they acted as the deployment vehicle for metadata; for example, you could copy an AOD file from the source system to the target system. Second, they provided runtime storage for model elements. The AOD file provided the physical storage for a native indexed sequential access method (ISAM) database that contained the metadata, and the runtime read model elements from this storage.

This method of managing metadata artifacts was not optimal. From a deployment perspective, AOD files didn't allow side-by-side installation of metadata in the same layer, didn't contain any structured information about their contents, and couldn't be digitally signed. From a runtime perspective, the AOD format supported only one table and provided no capability for adding or changing columns. To support the evolution of runtime scenarios, the product had to move toward a relationally correct schema.

In Microsoft Dynamics AX 2012, metadata is stored in the Microsoft SQL Server database along with business data. The tables containing the metadata are called the *model store*. This change removes all obstacles to providing a relationally correct schema. Elements, such as classes, tables, forms, methods, and controls, in the model store are grouped into models. Each model can be exported to a file-based format with the extension *.axmodel*. These files are managed assemblies and therefore, support digital signing, which makes them tamper-proof. Model files are the primary deployment vehicle for model elements in Microsoft Dynamics AX 2012.

In previous versions of Microsoft Dynamics AX, independent software vendors (ISVs) sometimes delivered textual source code files (in XPO format) to customers. Releasing source code files is an undesirable practice, but it was used to overcome restrictions on element IDs for combining multiple solutions in the same layer. With Microsoft Dynamics AX 2012, you no longer need to release source code files to customers. Model files do not contain element IDs, you can install multiple models in the same layer, and each model file contains a manifest that describes the model.

Note XPO files are still fully supported in Microsoft Dynamics AX 2012. Developers primarily use them to exchange source code and for storage in a version control system.

In previous versions of Microsoft Dynamics AX, a complete set of AOD files could be deployed in one operation, typically when a solution was moved from a staging system to a production system. The primary concern in this scenario is to reduce downtime. Copying all AOD files in one operation reduced downtime because there was no need to regenerate the Application Object Index (AOI) file or to recompile the application code. To satisfy the same need in Microsoft Dynamics AX 2012, you can export an entire model store in a binary file with the extension *.axmodelstore*. This file can be imported into the target system, and the system's downtime is restricted to the time it takes to restart the Application Object Server (AOS).

Layers

The Microsoft Dynamics AX 2012 runtime executes a program defined in the MorphX development environment. The program itself is defined as elements.

Unlike most systems, Microsoft Dynamics AX can contain multiple definitions of each element; for example, multiple implementations of the same method. These element definitions are stacked in layers. The Microsoft Dynamic AX runtime uses the element definitions from the highest layer in which they are found. For example, a method defined in the SYS layer (the lowest layer) is not used if another definition of the same method exists in any other layer.

This layered development approach provides several benefits, the most important being the ability to customize the program shipped by Microsoft, Microsoft partners, and ISVs without editing the original source code.

Note The layer metaphor is also used for graphical drawing tools. With layering, you can draw on top of an existing image without touching the original image underneath. Layers in Microsoft Dynamics AX work the same way, but with code and properties instead of pixels, shapes, and shades.

When you start Microsoft Dynamics AX 2012, you specify which layer you want to start in. By default, you start in the *USR* layer. Any element you create or edit is stored in that layer. If you edit an

element that exists in a lower layer, a copy of the element with your edits is moved to your layer. This process is known as *over-layering*.

Other benefits of layers include the ability to revert to the original definition of an element by deleting the over-layering version. You can also compare versions of an element; for example, to see which lines of code you have inserted. This is particularly useful during upgrade.

Caution Develop your solution one layer at a time, from the bottom up. Working in multiple layers at the same time on the same AOS is highly discouraged—even for different users. For more information see the Microsoft Dynamics AX 2012 white paper "Developing Solutions in a Shared AOS Development Environment" (*http://www.microsoft.com/download/en/details.aspx?id=26919*).

The process of editing an element from a higher layer than the current layer is known as under-layering. By design, these edits are routed to the higher layer.

Microsoft Dynamics AX 2012 has 16 metadata layers, each with its own purpose. Table 21-1 describes these layers in alphabetical order.

TABLE 21-1 Metadata layers.

Name	Description
USP (topmost)	Patch layer for the *USR* layer.
USR	User layer. This layer is intended for minor final customizations by power users of the system. These might include simple changes to the layout of a form and new security roles and tasks defined for the company.
CUP	Patch layer for the *CUS* layer.
CUS	Customer layer. This layer enables customer-specific customizations and extensions to the solution. This layer is typically developed by a Microsoft partner or an in-house development team.
VAP	Patch layer for the *VAR* layer.
VAR	Value-added retailer layer. Microsoft partners use this later to deliver customizations and extensions that typically are installed by multiple customers.
ISP	Patch layer for the *ISV* layer.
ISV	Independent Software Vendor layer. Registered Microsoft Dynamics AX ISVs can deliver solutions in this layer.
SLP	Patch layer for the *SLN* layer.
SLN	Solution layer. This layer contains vertical solutions provided by Microsoft partners.
FPP	Patch layer for the *FPK* layer.

Name	Description
FPK	Feature pack layer. This layer contains industry solutions provided by Microsoft. Public sector, Process industries, Retail, and Service industries are available in this layer.
GLP	Patch layer for the *GLS* layer.
GLS	Global solution layer. This layer contains regional extensions to the horizontal solution provided in the *SYS* layer.
SYP	Patch layer for the *SYS* layer.
SYS (Lowest)	System layer. Microsoft platform and foundation layer. This contains the horizontal application developed by Microsoft.

Within a layer, metadata elements are grouped into models. Models are covered in the following section.

Models

A model is a logical container of metadata elements, such as forms, tables, reports, and methods. For more information about element types, see Chapter 1, "Architectural overview."

The model store can contain as many models as you want. Figure 21-1 shows the relationship between layers, models, and elements.

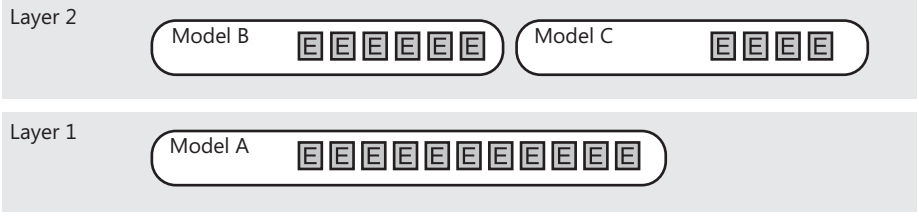


FIGURE 21-1 Layers, models, and elements.

Note The term *model* was selected for several reasons. First, any solution built in Microsoft Dynamics AX is a model of a real-life business. Second, a model is irreducible—even a part of a model is a model, so the term covers stand-alone solutions, extensions, customizations, patches, and so on. And finally; the term is simple and catchy—it will quickly become a part of your Microsoft Dynamics AX vocabulary.

You can have as many models in each layer as you want. This means you can segment your layer into as many models as you like. Here are some development scenarios where this can be useful:

- **When you deliver more than one solution** You can have a model for each solution you are working on. This enables you to work on them simultaneously.

- **When your solution is getting too large** You can segment your solution into several models and have each team or team member work on a different one. A model can be either self-contained or have dependencies on other models. This enables you to clearly define responsibilities between the model clearly, define the application programming interfaces (APIs) between the models clearly, build the models individually, and so on.
- **When you write unit tests** You can have a model for your production code and a model for your unit tests. These enable you to import all your unit tests, run them, and remove them from the system easily.

You can get a model in two ways: you can either create one on your own, or you can receive one from someone else. Because you can have as many models as you want in each layer, you can deploy models from several sources in the same layer.

Suppose you are a customer and want to install two ISV solutions that are available in the *ISV* layer. In previous versions, you would have had a tough choice to make. Either you picked your favorite solution and learned to live without the other one, or you invested in having the two solutions merged into one layer. This merge was technically challenging and costly if updates to either solution were released. In Microsoft Dynamics AX 2012, you just download the two models and then use AXUtil, a command-line utility that is available when you install Microsoft Dynamics AX, to import them. When a new version of either model is released, you simply use AXUtil to update the model.

The layer model element containment hierarchy has one restraint: an element can be defined only once per layer. In other words, you cannot install two models containing a definition of the same element in the same layer on the same system. Here are some examples:

- Two models that contain a class named *MyClass* cannot be installed side by side in the same layer.
- Two elements of the same type under the same parent element (or without a parent) cannot coexist in the same layer if they have same name or the same ID. For example, a table cannot have two fields with the same name or two fields with the same ID, and you cannot have two display menu items with the same name in the same layer.

This limitation enables the Microsoft Dynamics AX runtime to select the right version of an element to execute based on the layer in which it's contained.

You can encounter this limitation in two ways:

- You create an element and accidentally give it a name that is being used for another element in another model. A good way to avoid this is to prefix your new elements with a short string that uniquely identifies you, your company, or your solution. This practice is widely used.
- You customize an existing element that also has been customized by someone else in another model. There are various ways to limit the number of customized elements, such as by using events, but in some situations this is unavoidable.

Note Because element IDs are assigned at deployment time, the system automatically avoids duplicate IDs. Element IDs are covered next.

Element IDs

All element types have names, and a few element types also have an integer-based ID. The ID is a 32-bit integer and is assigned at installation time. This means that the same element might have a different ID on two different systems.

Note In previous versions of Microsoft Dynamics AX, IDs were 16-bit integers that were assigned at creation time from a pool of IDs for each layer. This could result in running out of IDs and ID collisions when installing solutions developed independently.

Two new properties have been introduced to support scenarios in which elements are upgraded or renamed:

- The *LegacyID* property has been added to the few element types that have an element ID. This property enables elements to keep their IDs from the AOD files when imported as a model file to a model store.
- The *Origin* property is a globally unique identifier (GUID) that uniquely identifies the element and eliminates the risk of a collision. The *Origin* property has been added to all root element types and element types with IDs. This property enables AXUtil (and other components) to recognize renamed elements during import.

AXUtil assigns element IDs when a model is imported, based on the following rules:

1. If an element already exists with the same *Origin*, replace the element and reuse its ID, else proceed to step 2.
2. If an element already exists with the same *Type*, *Name*, and *ParentID*, replace the element and reuse its ID, else proceed to step 3.
3. If the imported element has a *LegacyID*, and the *LegacyID* is available on the target system, add the element setting the ID to equal the *LegacyID*, else proceed to step 4.
4. Assign a new installation-specific ID that does not collide with any *LegacyID*s (greater than 60,000 for fields, and greater than 1,000,000 for all other element types).

This algorithm ensures that IDs are maintained in simple and advanced import scenarios. Consider a scenario where you have delivered several variations of the same solution to multiple customers as AOD or XPO files in Microsoft Dynamics AX 2009. This means that you probably maintain a copy of the source code for each of your customers in order to service them. As the number of customers

grows, so does the incentive to consolidate the variations into one common solution. Step 2 in the algorithm ensures that IDs are maintained on the customer's installation when the customer upgrades from a specialized solution to a common solution.

During regular development, the system maintains IDs, and you do not need to be concerned with them. However, you still need to pay attention to IDs in two situations: when upgrading a model and when moving from test to production. These two situations are covered in the following sections.

Note The data export/import functionality available under System administration automatically adjusts element ID references in the imported business data to match the element IDs on the target system. This adjustment skips all unstructured data. If you need to reference an element in a persisted container, for example, it is a best practice to reference the element by name.

Create a model

Before you implement your solution, you need to create a new model. You can create a model in several ways. You can do so in MorphX through Tools > Model Management > Create model, you can use Windows PowerShell from the Microsoft Dynamics AX 2012 Management Shell, or you can use AXUtil. The examples in this chapter use the latter.

```
AXUtil create /model:"My Model" /Layer:USR
```

Notice that you have to specify which layer the model belongs to. A model cannot span layers.

Note Each layer has a system-defined model. If you don't create your own model, the system-defined model is used. The system-defined model has certain deployment restrictions because its manifest is read-only. It is highly recommended that you create your own models.

After creating the model, you need to select it. In the status bar in MorphX, you can see the current model. You can change the model by clicking it. All elements that you create in the AOT are contained in the current model.

You can easily see which model an element belongs to by inspecting the element's properties. You can also enable a model indicator in the AOT on each element in Tools > Options > Development. You can move an element between models in the same layer by right-clicking the element, and then clicking Move to model.

Now that you have your model, you are ready to implement your solution.

Note You can delete any model by using the AXUtil delete command. This applies to models you have created and those you have installed. By using the */layer:<layer>* option, you can even delete all models in a layer.

Prepare a model for publication

When your implementation is complete, it is time to prepare the model for publication. But before you do, you may want to create a MorphX project that contains the elements in your model. You can create the project by using Tools > Model Management > Create project. This allows you to ensure that the model contains what you expect before you publish it. You can also use AXUtil to list the elements in a model.

```
AXUtil view /model:"My Model" /verbose
```

Preparing your model for publication consists of the following steps:

1. Set the model manifest.
2. Export the model to disk.
3. Add a digital signature.

Set the model manifest

A model is the container for your solution. You can describe your model in the model manifest. Table 21-2 contains a description of the properties of the model manifest. When you export your model, the exported file contains the manifest. The manifest helps consumers of your model understand its contents before installing it.

The simplest way to edit a manifest is to use XML notation:

```
AXUtil manifest /model:"My Model" /xml >MyManifest.xml
notepad MyManifest.xml
AXUtil edit /model: "My Model" @MyManifest.xml
```

TABLE 21-2 Model manifest properties.

Type	Description
Name	The full name of the model. The name often contains multiple words and typically the same name as used in marketing materials and other public documents.
Publisher	Publisher of the model; for example, "Microsoft Corp." The *Name* and *Publisher* properties must constitute a unique key. In other words, you cannot install two models with the same *Name* and *Publisher* in the same model store.
Description	A longer text string describing the model.
Display Name	A friendly name that is shown in the development environment, including in the status bar in MorphX and in the AOT. The *Display Name* is typically significantly shorter than *Name*, and it is often just a mnemonic value.
Version	A four-part version number; for example, 1.0.0.0.
Category	The category of the model. Models are grouped into four categories: ■ *Standard* A regular model. ■ *Hotfix* A model that contains a fix for an issue in another model. Hotfixes are typically delivered in a patch layer. ■ *Virtual* A model that is automatically created as a result of conflicting elements during model import, when using the */conflict:push* option. ■ *Temporary* A model that is used during the import process. At the end of the import, it will be deleted again. In most situations, you should set this property to *Standard*.
Details	The extension point of the manifest. If you need to capture more details about your model, you can place it here. The *Details* property must contain well-formed XML text. The model store and the Microsoft Dynamics AX runtime do not use this property. *Standard* models from Microsoft leave this property empty; *Hotfix* models include information about knowledge base (KB) articles in this property.

Note It is not possible to express dependencies between models in the model manifest. However, if you use the slipstream installation mechanism of the Microsoft Dynamics AX Setup program, you can control the installation sequence.

Export the model

When the manifest of the model is populated it is time to export the model to disk so that you can share it outside your organization:

```
AXUtil export /model:"My Model" /file:MyModel.axmodel
```

The extension *.axmodel* is used for model files. The model file contains all of the elements in the model, plus the model manifest. Model files are element ID–agnostic. When the elements in the model file are imported into a target system, they are assigned new installation-specific IDs. XPO files handle element IDs similarly.

Tip Under the covers, a model file is a managed assembly. This means you can use assembly reflection tools, like *ildasm*, to inspect the contents.

You can verify the model file contents using AXUtil:

```
AXUtil view /file:MyModel.axmodel /verbose
```

Note AXUtil is a powerful tool, and, for a command-line tool, also quite user friendly. Notice that some commands, like view and manifest, can be used either on a model in the model store and on a model file on disk. The most frequently used parameter is the */model* parameter. In the examples in this chapter, the name of the models are provided when using this parameter, but you can also specify the model ID (which typically is much shorter, and thus more convenient to write), or the model's manifest XML file. This latter option is particular useful when writing command scripts, such as build scripts for version control. All commands also supports a */verbose* parameter, which displays additional details about the command execution. For a complete list of commands and options, try *AXUtil /?*.

Sign the model

The model file is now ready to be shared. However, you should consider one more thing before making it publicly available. The model file contains binary code, and as such, this code can potentially harm a system, especially if the code is tampered with after it leaves your hands. To ensure that the customers who receive your model file can trust the file—or at least be able to tell that the model comes from a trustworthy source—you can add a digital signature to the model file.

When a signed model is imported, you are guaranteed the model file hasn't been tampered with since it was exported. If it has been tampered with, the import process fails. Microsoft Dynamics AX 2012 supports two ways of signing a model: strong name signing and Authenticode signing.

Strong name signing

To strong name sign a model, you need to use the .NET Framework Strong Name Tool, SN.exe, to generate a key/pair file. When you export your model to an *.axmodel* file, you specify the key to sign the model with.

```
SN -k mykey.snk
AXUtil export /model:"My Model" /file:MyModel.axmodel /key:mykey.snk
```

Authenticode signing

If you are a publisher of models, such as an ISV that provides models for download, consider Authenticode signing your model. If you do, your customers are guaranteed that the file hasn't been tampered with and that you created the model.

When an Authenticode signed model is imported, the model's publisher is authenticated. This means the model file can be traced to you.

To Authenticode sign a model file, first export it by using AXUtil. Then you use the SignTool to perform the actual signing:

```
signtool sign /f mycertprivate.pfx /p password MyModel.axmodel
```

Import model files

If you have received or downloaded a model file, you can import it by using AXUtil. The model file is always imported into the layer it was exported from. It is a best practice to stop the AOS before importing model files.

```
AXUtil import /file:SomeModel.axmodel
```

Note You don't have to specify file extensions when using AXUtil. The tool automatically adds the right extension if it is omitted. In this book, extensions are included for clarity.

Figure 21-2 shows a model that has been successfully imported into a layer in which a model already exists.

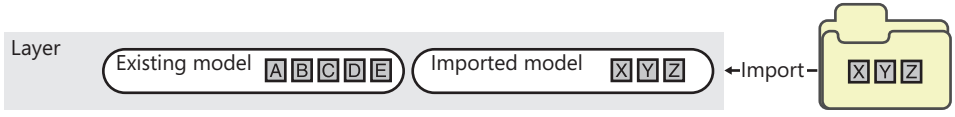


FIGURE 21-2 Side-by-side installation of two models.

The import operation will be cancelled if one or more elements from the model file are already defined in the layer into which the model is being imported. If you rerun the import operation with the */verbose* option, you will get a list of conflicting elements.

```
AXUtil import /file:SomeModel.axmodel /verbose
```

You have two options for proceeding with the import: *overwrite* and *push*.

Import model files with the *overwrite* option

You can decide to overwrite existing conflicting elements with the new definitions of the model element from the model file. You do so by specifying the */conflict:overwrite* option on the import command:

```
AXUtil import /file:SomeModel.axmodel /conflict:overwrite
```

Figure 21-3 shows the result of a successful import using the */conflict:overwrite* option. The imported model and the existing model that contained conflicting elements are linked after this operation. The models are linked because the existing model now is partial. The linkage prevents the imported model from being uninstalled unless the existing model is also uninstalled. This option is primarily used when delivering cumulative patches or service packs.

FIGURE 21-3 Side-by-side installation of two models using the */conflict:overwrite* option.

Import model files with the *push* option

The most typical solution to solve conflicts is the */conflict:push* option. This option creates a new virtual model in a higher layer containing the conflicting elements:

```
AXUtil import /file:SomeModel.axmodel /conflict:push
```

Figure 21-4 shows the result of a successful import operation using the */conflict:push* option. The elements in the virtual model are identical to those imported. In other words, existing models are not affected. After importing the model, log in to the layer containing the virtual model to resolve the conflict. You can use the compare functionality in the AOT to compare the conflicting versions of each element and resolve the conflict.

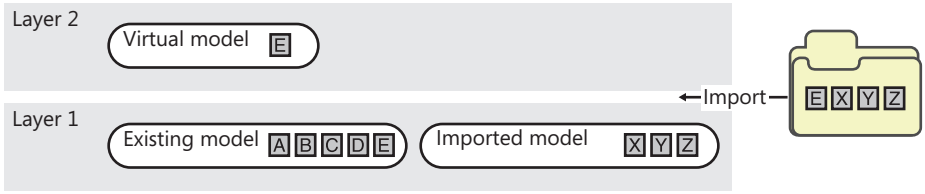


FIGURE 21-4 The result of an import operation using the */conflict:push* option.

If you resolve all conflicts in the same layer, there is no risk of running out of layers when using the */conflict:push* option. However, you may need to move the resolved elements into the same layer manually. For example, if you import a third model that conflicts with elements in the virtual model in Figure 21-4, the resulting virtual model will be created in Layer 3. After you resolve the conflicts in Layer 3, move the elements in Layer 3 to Layer 2. The easiest way to accomplish this is by exporting the elements from Layer 3 to an XPO file, deleting them, and importing them into Layer 2.

By default, the virtual model is created in the layer just above the layer the model is imported into. If you don't have developer access to that layer, you can force AXUtil to create the virtual model in a different layer (for example the *USR* layer) by using the */targetlayer* option, as shown in the following example:

```
AXUtil import /file:SomeModel.axmodel /conflict:push /targetlayer:USR
```

Upgrade a model

When you receive a newer version of a model and you want to replace the older version in the model store, it is important that you import the new model on top of the existing model. AXUtil automatically detects that the model already exists in the model store and performs the actions that are required to ensure the consistency of the model store:

```
AXUtil import /file:NewerModel.axmodel
```

By default, AXUtil import enters upgrade mode when a model with the same name and publisher already exists. Sometimes a model might be renamed or replaced by multiple new models (as the result of segmentation work, for example), or multiple models might be merged into one consolidated model. AXUtil supports upgrading existing models with new models. You can force AXUtil to use this mode by listing the files and models to upgrade, separated by a comma:

```
AXUtil import /file:f1,f2,f3 /replace:m1,m2,m3,m4,m5
```

Caution You might be tempted to uninstall an existing model before importing a newer version of the model, but if you do so, AXUtil does not enter upgrade mode and it assigns new element IDs to all elements being imported. This results in data corruption because business data contains the original element IDs. All references to elements in the uninstalled model will break. For more information, see the "Element IDs" section earlier in this chapter.

Move a model from test to production

It is a good practice to have a test or staging environment where changes to the system are prepared and tested before being deployed to a live production environment.

The model store provides features that you can use to export all model store metadata to a binary file and import it into a target system. Doing this creates a binary, identical copy of metadata between the two systems, including element IDs. Model store files have the extension *.axmodelstore*. Besides the metadata, the model store files also contain the compiled pcode and common intermediate language (CIL) code. This means that you do not have to compile the target system.

Note The size of model store files depend on the contents of the model store. A model store file for the standard installation of Microsoft Dynamics AX 2012 is about 2 GB. Model store files compress well, typically over 80 percent, and thus can be used as a simple backup.

Figure 21-5 shows the cleanest way of creating and preparing a test environment and deploying it to production. Variations and post-setup tasks to this process exist. For a thorough description, see the Microsoft Dynamics AX 2012 white paper "Deploying Customizations Across Microsoft Dynamics AX 2012 Environments" (*http://www.microsoft.com/download/en/details.aspx?id=26571*).

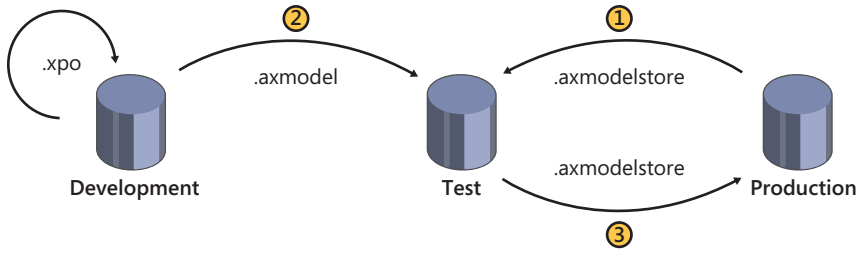


FIGURE 21-5 Creating and preparing a test environment and deploying a model store to production.

Note XPO files are not used in the process of deploying a system from test to production. They are mentioned in Figure 21-5 to show the scenarios that the three file formats should be used in. XPO files should be used for sharing source code between developers.

Create a test environment

The goal of creating a test environment is to ensure that the metadata in the model store is identical to the metadata in the model store in the production environment. The simplest way to achieve the goal is to create a new installation of Microsoft Dynamics AX, and then move the metadata from production to test. To move the metadata, you first need to export the model store from the production environment:

```
AXUtil exportstore /file:ProductionStore.axmodelstore
```

On the test system, you stop the AOS and then import the model store file:

```
Net stop AOS60$01
AXUtil importstore /file:ProductionStore.axmodelstore
Net start AOS60$01
```

Prepare the test environment

The goal of preparing the test environment is to update the system with new metadata, typically by installing new models or upgrading existing models. You import or upgrade models as explained earlier in this chapter.

After you import the models, start the Microsoft Dynamics AX client and complete the installation checklist. The most important steps are the compilation to pcode and CIL, because the products of these steps are part of the model store.

Extensive validation of the system is also recommended. Ensure that you validate both that the new functionality behaves as expected and that existing functionality hasn't regressed.

Deploy the model to production

The goal of deploying to production is to ensure that the metadata on the production system is updated with the metadata from the test environment. To move the metadata, you first need to export the model store from the test environment:

```
AXUtil exportstore /file:TestStore.axmodelstore
```

Import the model store file on the production system. To minimize downtime, AXUtil supports a two-phase import process. The first phase imports the metadata to a new schema in the database. This takes a few minutes and can occur while the production system is still live. The second phase replaces the model store metadata with the imported metadata from the schema. The takes a few seconds and must occur while the AOS is stopped.

Create a new schema:

```
AXUtil schema /schemaname:TransferSchema
```

Import the model store file into the new schema:

```
AXUtil importstore /file:TestStore.axmodelstore /schema:TransferSchema
```

When all users are logged off, stop the AOS:

```
Net stop AOS60$01
```

Apply the changes to the model store to move the new schema to the active schema:

```
AXUtil importstore /apply:TransferSchema /backupschema:dbo_backup
```

Restart the AOS:

```
Net start AOS60$01
```

Note Notice the use of the */backupschema* option in the example. With this option, you can quickly revert to the original metadata if unexpected issues arise. When you no longer need the backup schema, you can delete it by using the AXUtil *schema /drop:*<schemaname> command.

At this stage, the metadata in the production environment is identical to the metadata in the test environment. A few more tasks must be performed before the system is ready for users. These include synchronizing the database, creating role centers, deploying web content, setting up workflows, deploying cubes, importing integration ports, and deploying reports. For more information about these tasks, see the white paper "Deploying Customizations Across Microsoft Dynamics AX 2012 Environments" (*http://www.microsoft.com/download/en/details.aspx?id=26571)*.

Element ID considerations

Business data references metadata element IDs. The process outlined in the previous sections ensures that the element IDs in the production system remain unchanged, and thus ensures the integrity of the business data.

This is achieved by only exchanging metadata between test and production through model store files, which maintains the element IDs. For example, if the element IDs in the test environment and production environment are unsynchronized because XPO files or model files have been imported into both systems, you must rebuild the test environment.

The *importstore* command has a built-in safety mechanism. The command ensures that element IDs in the target system are identical to the element IDs in the file. If any conflicts are detected, the import operation stops. You can use the */verbose* option to get a list of the conflicts, and the */idconflict:overwrite* option to continue with the import operation anyway. Use the latter option only on a system where you don't care about the data — never in a production environment.

For more information, see the "Element IDs" section earlier in this chapter.

Model store API

The AXUtil utility used in all examples in this chapter provides a command-line interface to the model store commands offered by the model store API. A PowerShell interface is also available from the Microsoft Dynamics AX 2012 Management Shell.

Both these interface implementations use the managed assembly file *AXUtilLib.dll*. You can also use this assembly if you want to automate any model store operations. The assembly is referenced in X++, so you can easily access the model store API from X++. Some of the most common commands are available from the *SysModelStore* class.

The model store API also contains a method to generate license keys for a license code in the AOT based on the license holder name and serial number. The following example shows how to invoke this method from a managed website in an automated license purchasing scenario. For more information about how to protect your solution with a license code, see Chapter 14, "Customizing and extending Microsoft Dynamics AX."

```
using Microsoft.Dynamics.MorphX;
using Microsoft.Dynamics.AX.Framework.Tools.ModelManagement;

protected void Submit_Click(object sender, EventArgs e)
{
    string certPath = @"c:\Licenses\MyCertPrivate.pfx";
    string licensePath = @"c:\Licenses\" + Customer.Text + "-license.txt";
    string licenseCodeNameInAot = "MyLicenseCode";
    string certificatePassword = "password";  //TODO: Move to secure storage
    AXUtilContext context = new AXUtilContext();
    AXUtilConfiguration config = new AXUtilConfiguration();

    LicenseInfo licenseInfo = new LicenseInfo(licensePath, certPath,
                    licenseCodeNameInAot, Customer.Text, Serial.Text,
                    null, certificatePassword);
```

```
    config.LicenseInfo = licenseInfo;
    AXUtil AXUtil = new AXUtil(context, config);
    if (AXUtil.GenerateLicense())
    {
        Response.AddHeader("Content-Disposition",
                           "attachment;filename=license.txt");
        Response.TransmitFile(licensePath);
        Response.Flush();
        Response.End();
    }
}
```

APPENDIX

Resources for code upgrade

The resources listed in this section provide guidance and best practices for upgrading source code to Microsoft Dynamics AX 2012. Some of these resources contain information for system administrators, in addition to developers. Many of these resources are updated as new information becomes available or in response to customer requests, so check back often for the latest information.

Resource	Description and location
Upgrade to Microsoft Dynamics AX 2012	**Description:** Information for system administrators and developers that contains end-to-end instructions for upgrading to Microsoft Dynamics AX 2012 from Microsoft Dynamics AX 4.0 and Microsoft Dynamics AX 2009. **Location:** *http://technet.microsoft.com/en-us/library/dd362002*
Code Upgrade Tool User Guide	**Description:** Information about how to install and use the Code Upgrade Tool, which helps developers upgrade X++ code to Microsoft Dynamics AX 2012. **Location:** *http://technet.microsoft.com/en-us/library/hh535215.aspx*
Security Upgrade Advisor Tool User Guide	**Description:** Information about how to install and use the Microsoft Dynamics AX Security Advisor Tool, which simplifies the process of upgrading security settings from earlier versions of Microsoft Dynamics AX. **Location:** *http://technet.microsoft.com/EN-US/library/hh394895*
Upgrade Support for Managed Code	**Description:** Information about code upgrade tools and resources. **Location:** *http://msdn.microsoft.com/en-us/library/gg889224.aspx*
Upgrading Reports	**Description:** Information and guidance for upgrading reports to Microsoft Dynamics AX 2012 **Location:** *http://msdn.microsoft.com/en-us/library/gg724124*
Code Upgrade White Papers	**Description:** Several downloadable white papers covering numerous topics related to code upgrades. **Location:** *http://www.microsoft.com/en-us/download/details.aspx?id=20864*
Resource Page for Upgrading to Microsoft Dynamics AX 2012	**Description:** Downloads, hotfixes, troubleshooting tips, and additional resources related to upgrading. **Location:** *http://community.dynamics.com/product/ax/axtechnical/b/axresources/archive/2012/02/17/resource-page-for-upgrading-to-microsoft-dynamics-ax-2012.aspx*

Index

Symbols and Numbers

.chm files, 545
.NET AJAX, 222
.NET Business Connector
  Enterprise Portal architecture, 8, 197–198
  Enterprise Portal security, 232–235
  Enterprise Portal, developing for, 231–232
  proxies, Enterprise Portal, 226–228
.NET CIL (common intermediate language), compile and run X++, 126–128
.NET CLR interoperability statement, X++ syntax, 96
.NET Framework. *See also* Windows Workflow Foundation (WF)
  assemblies, hot-swapping, 84–85
  author managed code, 77–84
  chart control mark-up elements, 291–292
  EP Chart Control tool, overview, 289
  legacy systems, overview, 73
  plug-ins, developing, 7
  processing architecture, 4–6, 8
  third-party assemblies, use of, 73–76
.NET Framework Common Language Runtime (CLR), 13
.rsds (data connection file), PowerPivot, 336
.xpo files, create a build, 71

A

abstract, method modifier, 119
acceptance test driven development (ATDD), 535–536
access control. *See also security*
  form permissions, 356–359
  privileges, creating, 359–361
  security framework overview, 353–356
  security roles, privileges and duties, 361–362
access operators, X++ expressions, 95
accounting framework
  extensions, 662
  MorphX model element prefixes, 663
  overview, 659–662
  process states, 662–663
  when to use, 662
Accounting Journalization Rule, 660
Accounting Policy, 659
accounting, element naming prefix, 22
acknowledgement message, workflows, 260
action menu items, workflow artifacts, 266
Action pane
  controls, creating, 174–176
  details form design, 151–152
  Enterprise Portal, navigation form design, 156–157
  Enterprise Portal, web parts, 199
  model-driven list pages, creating, 218
  transaction details form design, 155
Action pane, list page design, 149
ActionMenuItemClicked, 212–213
ActionMenuItemClicking, 212–213
actions, element actions in AOT, 25–26
Active Directory
  Integrated Windows Authentication, 352–353
  security role assignments, 355
ActiveMode, 204
ActiveSectionIndex, 204
Activity, operations resource framework, 650–653
add-ins, Microsoft Office, 189–190
add-on functionalities
  delegates, X++ syntax, 120–122
  naming of, 21–22
address bar, navigation layer forms, 141
address book, 637
addresses, element prefix, 22
addRuntimeTask, 624
addTask, batch jobs, 624
aggregate, X++ select statements, 102
Aggregation property, associations, 48–49

AIF (Application Integration Framework)
custom services, creating, 388–391
overview, 386
send framework, 411–414
services, publishing, 400–401
AIF Document Service Wizard
artifacts, creating, 393
creating document services, 395–397
opening, 29
AifCollectionTypeAttribute, 391
AifDocumentService, 393
AifEntityKeys, 394
AJAX, Enterprise Portal development, 222
AllowDelete, AxGridView, 207
AllowEdit, AxGridView, 207
AllowFormCompanyChange, window type, 174
AllowGroupCollapse, AxGridView, 207
AllowGrouping, AxGridView, 207
AllowSelections, AxGridView, 207
ALM, Visual Studio, 534, 540–543
alternate keys
date-effective framework, 602–603
overview, 587–588
AmountMST, 316–317, 328–332
Analysis Currency, 329–332
Analysis Services Projects, modifying prebuilt projects, 319–323
AnalysisDimensionLabel, 326–327
AnalysisIdentifier, 326–327
AnalysisKeyAttributeLabel, 326–327
AnalysisMeasureGroupLabel, 326–327
analytics. *See also* Business Intelligence (BI)
analytic content, configuring, 310–311
Business Overview and KPI List web parts, 341–345
Excel reports, 340
overview, 333–335
presentation tools, choice of, 335
Report Builder, 346
SQL Server Power View, 335–340
Visual Studio tools, 346–349
anytype, 90, 93
AOD (Application Object Data) files, 687–688
AOS (Application Object Server)
architecture, 7
batch jobs, debugging, 629–630
configuration, performance and, 463–464
Help system, 549
processing architecture, 3–5, 7
AOSAuthorization, coding table permissions, 369–371
aosValidateDelete, 436
aosValidateInsert, 438–439
aosValidateRead, 433–434, 436
aosValidateUpdate, 433–434
AOT (Application Object Tree) modeling tool
AxDataSource, 203–204
creating elements, 23
element actions in, 25–26
element layers and models, 26
Enterprise Portal, developing for, 217
Jobs, overview, 88
modifying elements, 23–25
navigation in, 21–23
overview, 20
PageTitle, 200
processing architecture, 6–7
publishing services, 400–401
refreshing elements in, 25
security artifacts, developing, 356–363
Table Browser, overview, 52–53
third-party DLLs, referencing, 75–76
Toolbar, Enterprise Portal web part, 200
APIs (application programming interfaces)
AIF Send, 411–414
Batch API, 623–625
code access security (CAS), 124–126, 371–372
document services, overview, 392
Enterprise Portal architecture, 196–198
model store, 703–704
reflection, overview, 669–670
table data, reflection API, 673–676
treenodes, 680–685
API exceptions, dangerous, 41
application development environment, 6–7
application domain frameworks
accounting framework
extensions, 662
MorphX model element prefixes, 663
overview, 659–662
process states, 662–663
when to use, 662
dimension framework
constrain combinations of values, 656
create values, 656–657
extensions, 657
overview, 654–656
physical table references, 659
query data, 658–659

operations resource framework
extensions, 652–653
MorphX model element prefixes, 654
overview of, 648–652
when to use, 652
organization model framework
custom operating units, creating, 639–640
hierarchy designer, extending, 642
integration with other frameworks
application modules, 637–638
organization hierarchies, 635–637
organization types, 634–635
overview, 634
scenarios, modeling, 638–639
overview, 633–634
product model framework
extension of, 647–648
overview, 643–647
when to use, 647
source document framework
extensions, 666–667
MorphX model element prefixes, 667
overview, 664–665
when to use, 665
application frameworks, element naming prefix, 23
Application Integration Framework
custom services, creating, 388–391
overview, 386
send framework, 411–414
services, publishing, 400–401
application integration services, architecture, 8
application meta-model architecture
application data element types, 10–11
code element types, 13
documentation and resource element types, 16
license and configuration element types, 16–17
MorphX user interface control element types, 11–12
overview, 9
role-based security element types, 14
services element types, 13
web client element types, 14–15
workflow element types, 12–13
application models
creating models, 693–694
element IDs, 692–693
layers, 688–690
model store API, 703–704
models, overview, 690–692
moving from test to production, 700–703
overview, 687–688
publishing, preparing for, 694–699
upgrading, 699–700
Application Object Data (AOD) files, 687–688
Application Object Server (AOS)
architecture, 7
batch jobs, debugging, 629–630
configuration, performance and, 463–464
Enterprise Portal architecture, 197–198
Help system, 549
processing architecture, 3, 5, 7
Application Object Tree (AOT) modeling tool
AxDataSource, 203–204
creating elements, 23
element actions in, 25–26
element layers and models, 26
Enterprise Portal, developing for, 217
Jobs, overview, 88
modifying elements, 23–25
navigation in, 21–23
overview, 20
PageTitle, 200
processing architecture, 6–7
publishing services, 400–401
refreshing elements in, 25
security artifacts, developing, 356–363
Table Browser, overview, 52–53
third-party DLLs, referencing, 75–76
Toolbar, Enterprise Portal web part, 200
application platform, architecture of, 4–9
application programming interfaces (APIs)
AIF Send, 411–414
Batch API, 623–625
code access security (CAS), 124–126, 371–372
document services, overview, 392
Enterprise Portal architecture, 196–198
model store, 703–704
reflection, overview, 669–670
table data, reflection API, 673–676
treenodes, 680–685
ApplicationHelpOnTheWeb, 548
applications, processing architecture, 4–5
approvals, workflow artifacts, 264
approvals, workflow elements, 252
architecture
application platform, 6–9
Business Intelligence (BI), 299–300
Enterprise Portal, 196–198

architecture *(continued)*
five-layer solution architecture, overview, 4–6
overview, 3–4
report execution sequence, 278–279
security framework, overview, 351–356
service-oriented architecture, 386
workflow architecture, 256–262
area pages, designing, 144–146
arithmetic operators, X++ expressions, 95
artifacts, workflows, 264–265
ASP.NET
AxDataSource, 203–204
chart control mark-up elements, 291–292
datasets, Enterprise Portal, 201–203
Enterprise Portal architecture, 196–198
Enterprise Portal, AJAX, 222
EP Chart Control Tool, overview, 289
error handling, 231–232
GridView, 207
processing architecture, 4–6
UpdatePanel, 215
User control web part, 201
validation, 231
ViewState, Enterprise Portal, 228–229
assemblies, hot-swapping, 84–85
assert, code access security, 126
Asset, element prefix, 22
assignment statements, X++ syntax, 96
associated forms, permissions, 358
associations, metadata, 163
asynchronous mode, SysOperations, 468
ATDD (acceptance test driven development), 535–536
attributes
SysTest framework, new features, 527–533
UML associations, 48–49
X++ syntax, 123–124
Attributes, product model framework, 646
authentication
Enterprise Portal architecture, 197
Enterprise Portal, security, 232–235
models, signing, 696–697
security framework overview, 351–356
Authenticode, 696–697
authorization, security framework overview, 351–356
Auto variables, overview, 187
autogeneration, buttons, 220
auto-inference, form permissions, 356–359
AutoLookup, coding, 188–189
automated decisions, workflow elements, 253
automated tasks, workflow artifacts, 265
automated tasks, workflow elements, 253
AutoQuery, 170–172
autorefresh, 25
AutoSearch, metadata property, 170
avg
sample select statement code, 102
X++ select statements, 102
Ax, element prefix, 22
AxActionPanel, 211–212
AxaptaObjectAdapter, 204
AxBaseValidator, 231
AxBaseWebPart, 224–225
AxColumn, 205
AxCommon, 393–394
AxContentPanel, 215
AxContext, 223–225
AxContextMenu, 209
Axd documents
business document updates, 407–409
document services artifacts, 392–393
overview, 392
Axd, element naming prefix, 22
AxDataSource, 201–204, 225
AxDataSourceView, 203, 209
AxDatePicker, 216
AxDateSource, 219–221
AxDateTimeValueFormatter, 230
AxDateValueFormatter, 230
AxdDocument class, 393–394
AxdSend API, 412–414
AxEnumValueFormatter, 230
AxExceptionCategory, 231–232
AxFatal, exception handling, 231
AxFilter, 209–210
AxForm, 204, 231
AxFormPart, 216
AxGridView, 207, 210, 215, 231
AxGroup, 205–206, 215
AxGuidValueFormatter, 230
AxHierarchicalGridView, 208
AxInfoPart, 216
AxInternalBase, 395
AxLabel, 229–230
AxLookup, 210–211
axmodel, 687. *See also* model store
axmodelstore, 688. *See also* model store
AxMultiColumn, 205

AxMultiSection, 204
AxNumberValueFormatter, 230
AxPartContentArea, 216
AxPopup controls, 213–215
AxPopupBaseControl, 213–215
AxPopupChildControl, 213–215
AxPopupField, 214–215
AxPopupParentControl, 213–215
AxRealValueFormatter, 230
AxReportViewer, 216
AxSection, 204
AxStringValueFormatter, 230
AxTable class, 393–395
AxTableContext, 223–225
AxTableDataKey, 224–225
AxTimeValueFormatter, 230
AxToolbar, 212–213
AxToolbarButton, 212
AxToolBarMenu, 212
AxUpdate Portal, 239–240
AXUtil
    element IDs, 692
    importing models, 697–699
    model store API, 703–704
    models, upgrading, 699–700
AxValueFormatter, 230
AxValueFormatterFactory, 230
AxValueFormatValidator, 231
AxViewContext, 223–225
AxViewDataKey, 224–225

B

backing entity type, adding, 657
base enum elements, defined, 10
base enumeration types, 88, 93
Batch API, 623–625
Batch class, 49–50
batch framework
    Batch API, using, 623–625
    batch group, creating, 626–627
    batch jobs, creating, 618–625
    batch server, configuring, 625–626
    batch-executable class, creating, 616–617
    common uses of, 614–615
    debugging batch jobs, 629–631
    managing batch jobs, 628–629
    overview, 613–614
    performance and, 466–467, 615
Batch Job Form
    creating batch jobs, 619–622
    overview, 613
Batch Tasks form, 621–622
BatchHeader, 623–625
BatchRunable interface, 495–496
best practices
    exception handling, 105
    variable declaration syntax, 94
    X++ syntax, 93
Best Practices tool
    custom rules, 42–43
    errors and warnings, suppressing, 41–42
    overview, 20, 39–40
    rules, 40–41
    Trustworthy Computing, 372–373
    XML documentation, 116
BI. *See* Business Intelligence (BI)
bill of materials, element prefix, 22
binding
    AxToolbar, 212
    BoundField, 215
    chart controls, binding data series, 292–294
    chart controls, binding to dataset, 292
    control data binding, 173
    datasets, Enterprise Portal, 201
    field-bound controls, 178–179
    method-bound controls, 179
    object type, 92
bitwise operators, X++ expressions, 95
BOM, element prefix, 22
boolean
    value types, overview, 88
    variable declaration syntax, 93
BoundField, overview, 215
boxing, 110
break statement, X++ syntax, 96
Break, exception handling, 106
breakpoint
    AOS configuration, performance and, 463–464
    breakpoint statement, X++ syntax, 96
    shortcut keys, X++ code editor, 32
browsers, Enterprise Portal architecture, 197–198
buf2con, 425–426
built-in collection types, 88
built-in primitive type, 88
Business Connector, authoring managed code, 77–84
business documents, element prefix, 22

Business Intelligence (BI)
analytic content, configuring, 310–311
components of, 299–300
cubes, creating
generate and deploy cubes, 328–333
KPIs and calculations, adding, 333
metadata, defining, 325–328
requirements, identifying, 324–325
cubes, customizing, 311–319
cubes, extending, 319–323
customizing, overview, 309–310
displaying content in Role Centers
Business Overview and KPI List web parts, 341–345
Excel reports, 340
overview, 333–335
presentation tools, choice of, 335
Report Builder, 346
SQL Server Power View reports, 335–340
Visual Studio tools, 346–349
Enterprise Portal, web parts, 199
overview, 299
prebuilt BI solution, implementing, 301–309
properties, 326–327
Business Intelligence Development Studio, 323
business logic
overview, 188
workflows, creating, 264–265
Business Overview
Enterprise Portal, web parts, 199
Role Center displays, 341–345
business processes, defined, 245–246. *See also* workflow
business unit, defined, 635
buttons
action controls, creating, 174–176
action controls, overview, 175
AxToolbarButton, 212
details page, autogeneration, 220
Enterprise Portal, AJAX, 222
model-driven list pages, creating, 218–219
bytecode, overview, 87

C

C#, authoring managed code, 77–84
caching
cacheAddMethod, 418–419
CacheDataMethod, 419
CacheLookup, 421, 446–452
declarative display method caching, 419
elements, refreshing, 25
Enterprise Portal, developing for, 223
EntireTable cache, 453–454
indexing and, 421
labels, 230
MetadataCache, 225–226
number sequence caching, 465
performance
overview, 446
record caching, 446–452
unique index join cache, 452
RecordViewCache, 454–455
Server Configuration form, 463
SysGlobalCache, 456
SysGlobalObjectCache, 456
update conflicts, 409
CAL (client access license), 376–383
calculated measures, adding to cubes, 333
calendars, customizing cubes, 314–315
call stack, 88, 125–126
CallContext, consuming system services, 404–407
Called From, 420
camel casing, 93
cancel selection, shortcut key, 32
candidate key, alternate keys, 587–588
canGoBatchJournal, 617
canSubmitToWorkflow, 274
canSubmitToWorkflow, workflow artifacts, 266
Capability, 649–650
Caption property, forms, 160, 174
card sort, 145–146
CAS (code access security), 124–126, 371–372
casting statements, X++ syntax, 97
categories, product model framework, 646
changeCompany statement, X++ syntax, 98
ChangeGroupMode, 167
chart development tools, overview, 289
charts. *See* reporting
CIL (common intermediate language)
executing X++ as CIL, 466
troubleshooting, tracing, 487–488
claims-based authentication, 352–353
class element type, 13
class type, reference types, 89
Class Wizard, opening, 29
classDeclaration, attributes, 123–124

classes
attributes, X++ syntax, 123–124
fields, X++ syntax, 118
UML object models, 49–50
X++ syntax, overview, 117–118
classIdGet system function, reflection, 669, 672–673
ClearFieldValues, 214–215
client access license (CAL), 376–383
client access log, 490
client callbacks, eliminating, 424–425
client configuration, performance and, 464–465
client, method modifier, 119
Close button, autogeneration, 220
ClosePopup, 214–215
CLR (Common Language Runtime)
reference element types, 13
type conversions, 111
X++ interoperability, 108–112
CLRError, exception handling, 106
ClrObject, calling managed code, 76
code access security (CAS), 124–126, 371–372
code element types, overview, 13
code permission element type, 14
CodeAccessPermission, 125, 371–372
CodeAccessPermission.copy, 372
CodeAccessPermission.demand, 372
CodeAccessPermission.isSubsetOf, 372
CodeAccessSecurity, exception handling, 106, 372
coding. *See also* X++ programming language (code)
Auto variables, 187
business logic, 188
custom lookups, 188–189
form customization, overview, 184
method overrides, 184–186
collections list page example, 148–149
columns
AxColumn, 205
AxMultiColumn, 205
computed columns, creating views, 333
CombineXPOs, 71
CommandButton, 175
comments
inserting, shortcut key for, 32
X++ syntax, 115
common intermediate language (CIL)
executing X++ as CIL, 466
troubleshooting, tracing, 487–488
Common Language Runtime (CLR)
reference element types, 13
type conversions, 111
X++ interoperability, 108–112
common type, 90–91
Common, Area Page design, 145
Compare tool, 20, 54–59
compilation
Compiler output window, 38
compiling and running X++ as .NET CIL, 126–128
EB Web Applications, 220
errors, shortcut keys, 32
errors, use of semicolon, 95
intrinsic functions, reflection, 670–671
overview, 37–39
shortcut keys, X++ code editor, 32
Compiler tool, 20
compound statement, X++ syntax, 97
computed columns, views, 333
con2buf, 425–426
ConceptNum, 671
ConceptStr, 671
Concrete Source Documents, 664–665
conditional operators, X++ expressions, 95
conditions, workflow document classes, 268–270
configuration hierarchy, license codes, 378
configuration key element type, 16–17
configuration keys
Business Intelligence (BI), 309
license codes, 378–380
table inheritance hierarchy, 595
ConfigurationKey, temporary tables, 584
conflict resolution, project upgrades and, 29
Connect, Enterprise Portal web parts, 199
constrained table
data security policies, developing, 365–369
defined, 365
constructor encapsulation pattern, 129–130
constructor, defined, 120
consumer, eventing, 520
containers
converting table buffers, 425–426
variable declaration syntax, 93
content pane, navigation form design, 156–157
content pane, navigation layer forms, 142
ContentPage, window type, 174
context menu, AxContextMenu, 209
Context, Enterprise Portal development, 223–225
ContextMenuName, AxGridView, 207
context-sensitive Help topics, 561–562

ContextString, data security policies, 367
continue statement, X++ syntax, 97
controls
adding, overview, 172–173
control data binding, 173
Design node properties, 173–174
dialog box controls, 494
form permissions, 359
input controls, overview, 178–179
layout controls, overview, 176–178
ManagedHost control, 179–181
report elements, 282–285
runtime modifications, 174
conversationId, 412
CopyCallerQuery property, form queries, 172
Correction, date-effective tables, 605–606
CorrectPermissions, 360
COS, element prefix, 22
cost accounting, element prefix, 22
cost center, defined, 635
count, select statements, 102
Create New Document Service Wizard, 393
create, read, update, and delete (CRUD) permissions
forms, 356–359
menu items, 360
CreateNavigationPropertyMethods, 591–593
CreateNewTimePeriod, 605–606
CreatePermissions, 360
createRecord, 167
CRM, element prefix, 23
crossCompany, 99
CrossCompanyAutoQuery, metadata property, 169
Cross-reference tool
overview, 60–61
Cross-reference tool, overview, 20
CRUD (create, read, update, and delete) permissions
forms, 356–359
menu items, 360
cubes
calendars, customizing, 314–315
creating
generate and deploy cubes, 328–333
metadata, defining, 325–328
requirements, identifying, 324–325
currency conversion, 316–317
customizing, 311–319
extending, data source, 321
extending, DSV, 321
extending, external data sources, 322–323
extending, KPIs and calculations, 321
extending, measures and dimensions, 321
extending, overview, 319–321
financial dimensions, 313–314
languages, selecting, 315–316
SSAS deployment, 303–309
cues
cue element type, MorphX, 12
cue group element type, MorphX, 12
Cue Groups, creating, 218
CueGroupPartControl, 216
Enterprise Portal web part, 199
parts overview, 181
Role Center page design, 143
culture, Enterprise Portal formatting, 230
currency conversion
cubes, adding logic, 328–332
overview, 316–317
surrogate keys, 586–587
CurrentContextChanged, 224–225
CurrentContextProviderView, 224–225
CurrentDataAccess, 363
CurrentDate, time period filters, 345
Cust, element naming prefix, 22
custom rules, adding, 42–43
custom workflow providers, workflow artifacts, 266
customer contact, list page sample design, 146–148
Customer Group lookup, 210–211
customer relationship management, element prefix, 23
customers, element prefix, 22
customization
Business Intelligence (BI), 309–319
chart reporting, default override, 296
custom services, 388–391
delegates, X++ syntax, 120–122
deploying, 700
document services, overview, 392, 397–399
Enterprise Portal architecture, 196–198
model store API, 703–704
using code, overview, 184
web parts, 199–201

D

dangerous API exceptions, 41
data. *See also* metadata; also transaction performance
access to, Enterprise Portal, 225–226
Axd queries, creating, 395–396

chart controls, binding to, 292–294
consistency, date-effective framework, 604–606
control data binding, 173
cubes, metadata selection, 325–328
data contracts, custom services, 389–390
data contracts, X++ collections as, 391
extensible data security policies, creating, 364–369
external data source integration, 322–323
form data sources, 164–169
form method overrides, 185–186
metadata, form data source properties, 168–169
Microsoft Office add-ins, 189–190
report elements, design of, 282–285
security framework overview, 351–356
valid time state tables, use of, 362–363
data binding
BoundField, 215
datasets, Enterprise Portal, 201
Data DictionaryPerspectives, 325–326
data model, Cross-reference tool, 60
data object, document services, 393
data processing extensions, reports, 288
data source, element prefix, 22
data tier
architecture, 7
Business Intelligence (BI), 299–300
dataAreaId, indexing tips, 475–476
data-aware statements, X++ syntax, 99–104
database statements, X++ syntax, 99–104
database tier. *See also* transaction performance
alternate keys, 587–588
date-effective framework, 601–606
full-text support, 606–607
overview, 577
QueryFilter API, 607–612
surrogate keys, 585–587
table inheritance, 594–599
table relations, 588–593
temporary tables
creating, 583–585
InMemory tables, 578–582
TembDB temporary tables, 582–583
Unit of Work, 599–601
DataBound, AxGridView, 207
DataContractAttribute, 390, 495
DataKeyNames, AxForm, 204
DataMember, 203–204, 207
DataMemberAttribute, 390, 495
DataSet, details page, 219–221
datasets
AxGridView, 207
Enterprise Portal, 201–203
DataSetView, 204, 209, 225
DataSetViewRow, 204
DataSourceControl, 203–204
DataSourceID
AxDataSource, 203–204
AxForm, 204
AxGridView, 207
DataSourceName, Auto variables, 187
DataSourceView, 203
date-effective framework
data consistency, run-time support, 604–606
data retrieval support, 603–604
overview, 601
relational modeling, 601–603
dates
AxDatePicker, 216
calendars, customizing, 314–315
currency conversion logic, 330–332
DATE, 314
Date Dimensions, 314–315
date effectivity, overview, 168
date, value types, 88
date, variable declaration syntax, 94
valid time state tables, 355
DDEerror, exception handling, 106
Deadlock, 106
debugging
batch jobs, 629–631
Debug Target, 81–82
Debugger tool, MorphX, 20, 43–47
extensible data security policies, 368–369
managed code, 81–82
security, 373–375
Table Browser tool, 52–53
declarations, Help topics, 552–553
declarative display method caching, 419
decoupling, events, 521
Default Account, defined, 655–656
Default Dimensions, defined, 655
defaulting logic, document services, 399
delegate, eventing, 521–524
delegate, method modifier, 119
delegates, X++ syntax, 120–122
delete
current selection, shortcut keys, 32
InMemory temporary tables, 582
table relations, 588–593

delete permissions
forms, 356–359
menu items, 360
delete, deleting, and deleted methods, 168
delete_from, 104, 428, 435–436
DeletePermissions, 360
department, defined, 635
dependent workflow artifacts, 264–265
deployment. *See also* Microsoft Dynamics AX services
assemblies, hot-swapping, 84–85
cubes, 328–333
models, 700–703
third-party DLLs, 75–76
Version Control, 63–64
web part pages, 239–240
Derived Data Sources, 165–167
derived table, creating, 594
design definition, report elements, 282–285
Design node, control properties, 173–174
design patterns, X++, 128–133
details form
designing, 150–153
transaction details form, designing, 153–155
details page, creating, 219–221
DetailsFormMaster template, 161
DetailsFormTransaction template, 161
DeveloperDocumentation, 548
development environment. *See* application meta-model architecture; MorphX
development tools, element prefix, 23
device CAL, 376–383
dialog box controls, 494
Dialog template, 161
dictionary, reflection API, 670, 676–680
digital signature, assembly names, 74–75
Dimension Attribute
dimension framework overview, 654–656
query data, 658–659
Dimension Attribute Sets, defined, 655
Dimension Derivation Rule, 660
dimension framework
constrain combinations of values, 656
create values, 656–657
extensions, 657
overview of, 654–656
physical table references, 659
query data, 658–659
Dimension Sets, defined, 655
DimensionConstraintNode, 656
DimensionDefaultingService, 656–657
dimensions, cubes, 321
DimensionStorage, 656–657
DimensionValidation
validateByTree, 656
Dir, element prefix, 22
directory, element prefix, 22
DirPartyTable_FK, 592
DirPersonName, 601–603
disableCache, 450
display menu items, workflow artifacts, 266
display, method modifier, 119
DisplayGroupFieldName, AxGridView, 207
DLLs (dynamic-link libraries)
referencing managed DLLs, 75–76
using third-party assemblies, 73–76
do while statement, X++ syntax, 97
document class, workflows, 268–270
document files, Help server, 547–549
document head, Help content, 553–556
document services
artifacts, 392–393
AxdDocument class, 393–394
AxTable classes, 395
creating, 395–397
customizing, 397–399
overview, 392
document set, Help system, 571
documentation element type, 16
documents
element naming conventions, 22
processing architecture, 5
doDelete, 428
doInsert, 428
doUpdate, 428
DropDialog template, 161
DropDialogButton, 175
DSV, overview, 321
DuplicateKeyException, 107
DuplicateKeyException-NotRecovered, 107
duties, security framework overview, 353–356
duty element type, overview, 14
dynalinked data sources
LinkType, metadata property, 169
Microsoft Dynamics AX client, 164–169
Dynamic Account, defined, 656
dynamic role assignment, 355
dynamic-link libraries (DLLs)
referencing managed DLLs, 75–76
using third-party assemblies, 73–76

DynamicsAxEnterprisePortal, 237
DynamicsAxEnterprisePortalMOSS, 237
DynamicsAxWebParts, 237
DynamicsLeftNavProvider, 235
DynamicsMOSSTopNavProvider, 235
DynamicsSearch, 237
DynamicsTopNavProvider, 235

E

economic resources, element prefix, 22
EcoRes, element prefix, 22, 647–648
editing
    AllowEdit, AxGridView, 207
    details form design, 152
    edit permissions, menu items, 360
    edit, method modifier, 119
EDT. *See* extended data type (EDT)
EDT Relation Migration tool, 589
EffectiveAccess, form permissions, 358
EffectiveBased, date-effective tables, 605–606
element IDs
    application models, 692–693
    model deployment, 702–703
element, Auto variables, 187
elements
    actions in AOT, 25–26
    Compare tool, 54–59
    creating in AOT, 23
    Cross-reference tool, overview, 60–61
    history of, viewing, 69
    layers and models in AOT, 26
    modifying in AOT, 23–25
    naming conventions, 21–22
    pending, Version control, 70
    Projects, creating, 27
    property sheet, overview, 30–31
    refreshing elements, AOT, 25
    upgrades, conflict resolution, 29
    version control system life cycle, 64–65
    Version Control tool, overview, 62–64
    workflow elements, 252–253
Enable block selection, 32
encryption, Enterprise Portal, 235
endpoints, associations, 49
Enterprise Portal. *See also* Enterprise Portal, developing for
    AOT elements, 201
    architecture, 8, 196–198
    authentication, overview, 352–353
    AxActionPanel, 211–212
    AxColumn, 205
    AxContentPanel, 215
    AxContextMenu, 209
    AxDataSource, 203–204
    AxDatePicker, 216
    AxFilter, 209–210
    AxForm, 204
    AxFormPart, 216
    AxGridView, 207
    AxGroup, 205–206
    AxHierarchicalGridView, 208
    AxInfoPart, 216
    AxLookup, 210–211
    AxMultiColumn, 205
    AxMultiSection, 204
    AxPartContentArea, 216
    AxPopup controls, 213–215
    AxReportViewer, 216
    AxSection, 204
    AxToolbar, 212–213
    BoundField, 215
    charting controls
        chart development tools, 289
        data series, binding, 292–294
        EP Chart Control, creating, 290–291
        mark-up elements, 291–292
        overview, 289
    CueGroupPartControl, 216
    datasets, 201–203
    overview, 195
    presentation tier architecture, 8
    processing architecture, 5
    Visual Studio development environment, 7
    web client experience, designing, 155–157
    web parts, overview of, 199–201
    workflow menu items, 252
Enterprise Portal, developing for
    AJAX, 222
    Context, 223–225
    data, access, 225
    details page, creating, 219–221
    Enterprise Search, 240–243
    error handling, 231–232
    formatting, 230
    labels, 229–230
    metadata, access to, 225–226
    model-driven list page, creating, 217–219
    overview, 216–217

Enterprise Portal, developing for, *(continued)*
proxy classes, 226–228
security, 232–235
session disposal and caching, 223
SharePoint integration
Enterprise Search, 240–243
site definitions, page templates, and web parts, 237–239
site navigation, 235–236
themes, 243
web part page, import and deploy, 239–240
validation, 231
ViewState, 228–229
Enterprise Search
Enterprise Portal and SharePoint integration, 240–243
EntireTable caching, 421, 453–454, 463
entity relationship data model, 51
enumeration types, 88
EPApplicationProxies, 226–228
EP Chart Control
creating, 290–291
data series binding, 292–294
mark-up elements, 291–292
overview, 289
EPSetupParams, 237
EP User Control with Form template, 219–221
EP Web Application Project template, 219–221
Error, exception handling, 107
errors
compiler errors, 37–39
error handling, Enterprise Portal development, 231–232
error warnings, Enterprise Portal web parts, 200
suppressing, Best Practices tool, 41–42
Esc key, 32
evalBuf, code access security, 124–126
event handlers
InMemory temporary tables, 582
pre- and post-, X++ syntax, 122–123
pre- and post-events, overview, 522–523
workflow artifacts, 266
workflows, 252
workflows, state management, 266–267
event logging, processing architecture, 5
event payload, defined, 521
eventhandler, delegate subscription, 121–122
eventing extensibility pattern
delegates, 521–522
event handlers, 523–524
eventing example, 524–525
overview, 520–521
pre and post events, 522–523
Excel
architecture, 9
authoring managed code example, 77–84
PowerPivot, SQL Server Power View, 335–340
reports, displaying in Role Centers, 340
templates, building, 190–193
Excel Services for SharePoint, report display, 340
Excel Services web part, report display, 340
exception handling, X++ syntax, 105–108
exceptions, dangerous APIs, 41
exchange rates, 329–332
Exchange Rates By Day, 331–332
execute current element, 32
execute editor script, 32
ExecutePermission, 371–372
executeQuery method, 172
ExecuteQuery, AxFilter, 209
ExistJoin, 164
exists, 103
exists method, 131–132
ExpansionColumnIndexesHidden, 207
ExpansionTooltip, AxGridView, 207
expense reports, 638
ExplicitCreate, 168
exporting elements, Compare tool, 56
exporting models, 695–696
expressions
expression builder, workflow document, 250, 269
X++ syntax, 95
extended data type (EDT)
AmountMST, currency conversions, 316–317
AxLookup, 210–211
defined, 10
overview, 88, 92–93
table relations, 588–590
value types, overview, 88
variable declaration syntax, 94
extensible data security framework (XDS)
debugging data security policies, 368–369
organization model framework integration, 638
policies, creating, 364–369
temporary table constructs, 368
extension framework
creating extensions, 517–518
extension example, 518–520
extensions, SQL Server Reporting Services (SSRS), 8

external name, service contracts, 389–390
ExternalContextChanged, 224–225
ExternalContextProviderView, 224–225

F

F1, context-sensitive Help, 561–562
FactBox
  AxPartContentArea, 216
  details form design, 152
  Enterprise Portal, navigation form design, 156–157
  form parts, overview, 182
  list page design, 150
  model-driven list pages, creating, 218
  MorphX user interface control element type, 12
  navigation layer forms, 142
  transaction details form design, 155
FastTabs, details form design, 150–153
FastTabs, TabPage controls, 177–178
field level properties, 327–328
field lists, performance and, 456–462
field-bound controls, 178–179. *See also* controls
fields, Help topics, 559–561
fields, X++ syntax, 118
file transfers. *See also* .NET Framework
  legacy systems, overview, 73
FileIOPermission, 371–372
filters
  AxDataSource, 203–204
  AxFilter, 209–210
  Projects, automatic generation of, 28
  QueryFilter, 172, 607–612
  ResetFilter, 209
  ShowFilter, AxGridView, 208
  SystemFilter, 209
  SysTest framework, new features, 530–533
  time period, Business Overview web part, 344–345
  UserFilter, 209
final, method modifier, 119
financial dimensions, cubes, 313–314
financial dimensions, organization model framework, 637
find method, 131–132
Find tool, 20
  overview, 53–54
finished goods management. *See* product model framework
FireCurrentContextChanged, 224–225
FireExternalContextChanged, 224–225
firstFast, 99, 476
firstOnly, 100
firstOnly10, 100
firstOnly100, 100
firstOnly1000, 100
FISCALPERIODDATEDIMENSION, 314
flexible authentication, overview, 352–353
flush statement, X++ syntax, 97
folders
  Help content, publishing, 567–571
  Projects, automatic generation of, 27–29
for statement, X++ syntax, 97
forceLiterals, 100
forceNestedLoop, 100
forcePlaceholders, 100
forceSelectOrder, 100
foreign key relations, 590–593
foreign keys, surrogate, 168
form element type, MorphX, 12
form part element type, MorphX, 12
Form Parts, model-driven list pages, 218
form parts, overview, 182
form rendering, MorphX user interface controls, 11–12
Form.DataSources, 162
Form.Designs.Design, 162
Form.Parts, 162
formatting, Enterprise Portal, 230
FormDataSource.AutoQuery, 170
FormDataSource.cacheAddMethod, 418–419
FormDataSource.create method, 166
FormDataSource.init, 172
FormDataSource.query, 172
FormDataSource.queryRun.query, 172
forms. *See also* details form; also Enterprise Portal, developing for; also Purchase Order form; also Sales Order form; also user experience, designing
  debugging security, 373–375
  display and edit methods, cacheAddMethod, 418–419
  form metadata, 162–164
  form patterns, 160–162
  form queries, overview, 170–172
  method overrides, 184–186
  navigation layer forms, design basics, 141–142
  navigation layer forms, Enterprise Portal design, 156–157

forms *(continued)*
permissions, setting, 356–359
working with, overview, 159–160
FormTemplate, 218
forUpdate, 100, 446–452
Found, record caching, 446–452
FoundAndEmpty, record caching, 446–452
foundation application domain partition, 4–6
FutureDataAccess, 363

G

General Journal, 660
general ledger, element prefix, 23
generateOnly, 100, 368–369
generic group reference, 132–133
Generic Test Case, 541
GetClosePopupEventReference, 214–215
GetCurrent, 204
getDataContractInfo, 494
GetDataSet, 204
GetFieldValue, AxPopup, 214–215
GetOpenPopupEventReference, 213–215
getQueryName, 269
getServiceInfo, 494
getter methods, 592
global address book, element prefix, 22
Glossary, Help system, 548
Go to implementation, 32
Go to next method, 32
Grid control, overview of, 176, 178
GridColumnIndexesHidden, 207
GridView
AxGridView and, 207
BoundField, 215
group by, select statement code sample, 102
Group control, overview of, 176
GroupField, 207
GroupFieldDisplayName, 207
grouping, AxGridView, 207
GroupMask, 28–29
Groups, automatic project generation, 27–29
Guest accounts, Enterprise Portal, 232–235
GUID, AxGuidValueFormatter, 230

H

header view, transaction details form, 154
headers, XML insert shortcut key, 32
help documentation set element type, 16
Help system
creating content
add labels, fields, and menu items, 559–561
context-sensitive topics, 561–562
non-HTML topics, 565–567
overview, 550–552
table of contents, creating, 563–565
topics, create in HTML, 552–559
updating content, 562–563
Help server, 546–549
help text, label file, 229–230
help text, SysOperationHelpTextAttribute, 495
Help viewer, 546–547
Microsoft Dynamics AX client, 547
overview, 545–549
processing architecture, 8
publisher, 550
publishing content, 567–571
shortcut keys, X++ code editor, 32
summary page, 550–551
table of contents, 550
topics, 549–550
troubleshooting, 572–573
HiddenField, AxPopupField, 214–215
hierarchy
organizational model framework, 635–637
type hierarchies, 89–93
hierarchy designer, extending, 642
HierarchyIDFieldName, 208
HierarchyParentIdFieldName, 208
history, batch jobs, 629
horizontal application domain partition, 4–6
hot-swapping assemblies, 84–85, 463–464
HRM/HCM, element prefix, 22
human resources management processes
element prefix, 22
organization model framework integration, 638
human workflows, defined, 246–249. *See also* workflow
HyperLinkMenuItem, model-driven list pages, 218

I

ICustomFormatter, 230
identifierStr, 671
if statement, X++ syntax, 97
IFormatProvider, 230
IIS (Internet Information Services), 8, 197–198

implementation patterns, X++, 128–133
ImplicitCreate, 168
ImplicitInnerOuter, 167
importing elements
    Compare tool, 56
    web part pages, 239–240
importing models, 697–699
Included Columns, indexing tips, 475–476
incremental search, shortcut key, 32
index
    ActiveSectionIndex, 204
    alternate keys, 587–588
    caching and, 421
    database query sample, 101
    Enterprise Search, 240–243
    ExpansionColumnIndexesHidden, 207
    GridColumnIndexesHidden, 207
    inMemory temporary tables, 423
    performance tips, 475–476
    record caching, 448–449
    SelectedIndexChanged, 208
    unique index join cache, 452
indexed sequential access method (ISAM), 577–582
IndexTabs, TabPage controls, 176–178
industry application domain partition, 4–6
info part element type, MorphX, 12
Info Parts, model-driven list pages, 218
info parts, overview, 181
Infolog
    Enterprise Portal, web parts, 200
    validation, 231
information dissemination, events, 521
information messages, Enterprise Portal web parts, 200
infrastructure callback, workflow, 261
inheritance
    RunBase, 510–511
    tables, 91, 165–167, 594–599
inheritance, metadata, 163–164
init method, Enterprise Portal datasets, 202
initializing, SysListPageInteractionBase, 218
Inline, WindowMode, 220
InMemory temporary tables, 422–423, 428, 578–585
InnerJoin, 164
input controls, overview, 178–179. *See also* controls
Inquiries, Area Page design, 145
insert method, transaction statement, 104
insert_recordset
    RowCount, 104
    sample statement, 104
    table hierarchies and, 428
    transaction performance and, 429–432
    transferring code into set-based operations, 442–444
insertDatabase, transaction performance, 437–439
InsertIfEmpty, metadata property, 169
installation, third-party DLLs, 75–76
InstanceRelationType, 594
int
    value types, overview, 88
    variable declaration syntax, 94
int64
    value types, overview, 88
    variable declaration syntax, 94
Integrated Windows Authentication, 352–353
integration tier, Business Intelligence, 299–300
IntelliMorph, 8, 33
IntelliSense, 73–76
InteractionClass, 218–219
interactive functions, reports, 294–295
interfaces
    code access security (CAS), 124–126
    fields, syntax, 118
    label file, 229–230
    methods, syntax, 118–120
    reference types, overview, 89
    UML object models, 49–50
    X++ syntax, overview, 117–118
inter-form dynalinks, 165
Internal, exception handling, 107
International Financial Reporting Standards, segregation of duties, 354
Internet Information Services (IIS), 8, 197–198
InteropPermission, 371–372
intra-form dynalinks, 165
intrinsic function, reflection, 669–671
Invent, element prefix, 22
inventory management, 22, 465. *See also* product model framework
Inventory Model Group, 645
is operator, 92
ISAM (indexed sequential access method), 577–582
IsConsumingMemory, 684
IsGetNodeInLayerSupported, 684
IsLayerAware, 684
IsLookUp, 327

IsModelElement, 684
IsRootElement, 684
isTmp, 582
IsUtilElement, 684
IsVCSControllableElement, 684
Item Group, 645

J

JMG, element prefix, 23
job element types, 13
Jobs, overview of, 88
joined data sources, Microsoft Dynamics AX client, 164–169
joins
    LinkType, metadata property, 169
    maximum number, 463
    select statement code sample, 102
JoinSource, metadata property, 169
Journalization Processor, 660–662
Journalization Rules, 660–662
Journals, Area Page design, 145

K

keywords, data-aware statement options, 99
KM, element prefix, 23
knowledge management, element prefix, 23
KPI
    adding to KPI List web part, 342–344
    cubes, creating, 333
    SSAS projects, 321
KPI List web part, Role Center displays, 341–345

L

Label editor tool
    creating labels, 35–36
    overview, 20, 33–35
    referencing X++ labels, 36–37
    shortcut keys, X++ code editor, 32
labels
    Enterprise Portal, developing for, 229–230
    Help topics, 559–561
    label file element type, 16
    Label Files, 33–34
    Label Files Wizard, 35
language
    cubes, customizing, 315–316
    formatting, Enterprise Portal, 230
    label editor, 33
    label editor, creating labels, 36
layer comparison, Project development tools, 29
layers, metadata
    models, overview, 690–692
    overview, 688–690
layout controls
    input controls, overview, 178–179
    overview of, 176–178
layout, hiding report columns, 286–288
Ledger Dimensions
    creating, 656–657
    defined, 655
Ledger, element prefix, 23
Left navigation, Enterprise Portal web part, 200
legacy systems. *See also* .NET Framework
    legacy data source integration, 323
    working with, 73
LegacyID, 692
legal entity, defined, 634
license code element type, 16–17
license keys, generating, 703–704
licensing
    CALs, types of, 381–382
    customization and, 383
    overview, 376–383
line view, transaction details form, 154
line-item workflows, overview, 253
links
    data sources, Microsoft Dynamics AX client, 164–169
    Help content, 558
    hiding, ShowLink, 236
    HyperLinkMenuItem, 218
LinkType, 164, 169
list definition element type, 15
list pages
    designing, 146–150
    model-driven list page, creating, 217–219
    performance optimization, 476
ListPage template, 161
ListPage, window type, 174
ListPageInteraction, 218
literalStr, 36–37, 671
loadData, 653

local function statement, X++ syntax, 98
localization
label editor, 33
label editor, creating labels, 36
logging, InMemory temporary tables, 582
logical operators
logical approval and task workflows, 260–262
X++ expressions, 95
LogonAs, Enterprise Portal security, 232–235
LogonAsGuest, Enterprise Portal security, 232–235
lookups
AxLookup, 210–211
custom, coding, 188–189
table relations, 588–593
LookupType, 211
loop iterations, reducing, 477–478

M

macro element types, overview, 13
macros, X++ syntax, 113–115
Main Account Derivation Rule, 660
Main Account Dimension Attribute, 658–659
Main Account Dimension List Provider, 660
MainMenu, adding menus, 183. *See also* menus
MaintainUserLicense, 383
managed code
assemblies, hot-swapping, 84–85
authoring, 77–84
third-party assemblies, 76
ManagedHost control, 179–181
manifest, models, 694–695
manual decisions, workflow elements, 253
map element type, defined, 10
map type, reference types, 89
marketing management, element prefix, 23
master data sources, 164–169
master scheduling, performance and, 465
maxOf, X++ select statements, 102
measures, cubes, 321, 333
memory heap, defined, 88
MenuButton, 175
MenuItem
adding, 182
AxToolbar, 212
CopyCallerQuery, 172
overview, 175
MenuItemButton
CopyCallerQuery, 172
overview, 174–175
menus
adding, overview, 183
batch jobs, creating, 618
customizing, 383
Enterprise Portal, security, 233–235
form permissions, 360
Help topic labels, 561
Help topics, 559–561
menu definitions, 183
menu element type, MorphX, 11
menu item element type, MorphX, 11
model-driven list pages, creating, 218
SharePoint navigation, 236
web, AxActionPanel, 211–212
workflow artifacts, 266
workflow menu items, 252, 270
messages
Enterprise Portal, web parts, 200
send framework, 411–414
metadata
access to, Enterprise Portal, 225–226
associations, 163
cubes, metadata selection, 311–312, 325–328
custom service artifacts, 388
document service, artifacts, 392–393
element IDs, 692–693
form data source properties, 168–169
form metadata, Microsoft Dynamics AX client, 162–164
Help topic updates, 563
Help topics, 566–567
inheritance, 163–164
layers, 688–690
managing artifacts, overview, 687–688
model store API, 703–704
models
creating, 693–694
moving to production, 700–703
overview, 690–692
preparing for publication, 694–699
upgrading, 699–700
providers, workflow, 254–255
system services, 388
system services, consuming, 404–407
X++ collections as data contracts, 391
Metadata service, Enterprise Search, 240–243
MetadataCache, Enterprise Portal, 225–226
meta-model architecture. *See* application meta-model architecture
method invocations, X++ expressions, 95

method overrides, 184–186
method-bound controls, overview, 179
methods, X++ syntax, 118–120
Microsoft Dynamics AX Reporting Project, 282
Microsoft .NET Framework. *See* .NET Framework
Microsoft ASP.NET, processing architecture, 4–6
Microsoft Dynamics AX
  introduction to, 3–4
  model and application database, 7
Microsoft Dynamics AX Application Object Server (AOS). *See* AOS (Microsoft Dynamics AX Application Object Server)
Microsoft Dynamics AX Business Connector
  authoring managed code, 77–84
Microsoft Dynamics AX client
  action controls, 174–176
  Auto variables, 187
  business logic, 188
  control data binding, 173
  control overrides, 172–173
  controls, Design node properties, 173–174
  controls, run-time modifications, 174
  custom lookups, 188–189
  customizing forms, overview, 184
  Excel templates, building, 190–191
  form data sources, 164–169
  form metadata, 162–164
  form patterns, 160–162
  form queries, 170–172
  forms, overview, 159–160
  Help system interaction, 547
  input controls, 178–179
  layout controls, 176–178
  ManagedHost control, 179–181
  method overrides, 184–186
  Microsoft Office client integration, 189–193
  navigation items, overview, 182–183
  overview of, 159
  parts, overview, 181–182
  user templates, adding, 192–193
  Word templates, 191–192
Microsoft Dynamics AX Enterprise Portal. *See* Enterprise Portal
Microsoft Dynamics AX Report Definition Customization Extension (RDCE), 286–288
Microsoft Dynamics AX services
  asynchronous invocations, 409–411
  consuming
    business document updates, 407–409
    external web services, 414
    system services, 404–407
    WCF client sample, 402–404
  custom services, 388–391
  document services
    artifacts, 392–393
    AxdDocument class, 393–394
    AxTable classes, 395
    creating, 395–397
    customizing, 397–399
    overview, 392
  overview of, 385–387
  performance considerations, 415
  publishing, 400–401
  security, 400
  send framework, 411–414
  system services, 387–388
Microsoft Dynamics AX Trace Parser, 479–488
Microsoft Dynamics AX Windows, 3–4
Microsoft Dynamics Public, security, 232–235
Microsoft Excel. *See* Excel
Microsoft Office
  architecture, 4–6
  presentation tier architecture, 9
  processing architecture, 3–4
Microsoft Office client
  Excel templates, 190–191
  integration with, 189–193
Microsoft SharePoint Server. *See* SharePoint Server
Microsoft SQL Server. *See* SQL Server
Microsoft SQL Server Analysis Services. *See* SQL Server Analysis Services (SSAS)
Microsoft SQL Server Reporting Services. *See* SQL Server Reporting Services (SSRS)
Microsoft Technology platform, architecture overview, 4–6
Microsoft Test Manager
  acceptance test driven development (ATDD), 535–536
  ALM solution, 534
  ordered test suites, 539–540
  overview, 533–534
  shared steps, 536–539
  Team Foundation Build, 540–543
  test case evolution, 538
  test selection, 542–544
Microsoft Visio
  Reverse Engineering tool, MorphX, 47–51
Microsoft Visual SourceSafe (VSS), 62–64
Microsoft Visual Studio Integrated Development Environment (IDE), 4–6

Microsoft Visual Studio Team Foundation Server (TFS), 62–64
Microsoft Windows client, architecture, 4–6, 8
Microsoft Windows Server, architecture, 4–6
Microsoft Word client
  architecture, 9
  templates, building, 191–193
migration, EDT Relation Migration tool, 589
minOf, X++ select statements, 102
modal dialog settings, details page, 220–221
Modal, WindowMode, 221
Model Editor, production reports and, 282
Model Manifest, 694–695
model store
  APIs, 703–704
  element IDs, 692–693
  layers, overview, 688–690
  models, creating, 693–694
  models, overview, 690–692
  models, preparing for publication, 694–699
  moving from test to production, 700–703
  overview, 687–688
  upgrading models, 699–700
model-driven list page, creating, 217–219
Model-View-Controller (MVC), 494
Month_LastMonth, time period filters, 345
MorphX
  accounting framework, model element prefixes, 663
  Application Object Tree
    creating elements, 23
    element actions, 25–26
    element layers and models, 26
    modifying elements, 23–25
    navigation, 21–23
    refresh elements, 25
  Best Practices tool, overview, 39–43
  client-side reporting solutions, 276–277
  code compiler tool, overview, 37–39
  Compare tool, overview, 54–59
  Cross-reference tool, overview, 60–61
  datasets, creating, 201
  debugger tool, overview, 43–47
  Enterprise Portal architecture, 196–198
  Enterprise Portal, developing for, 216
  Find tool, overview, 53–54
  Label editor, overview, 33–37
  models, preparing for publication, 694–699
  operations resource framework, element prefixes, 654
  overview of, 6–7, 19–21
  processing architecture, 5, 7–9
  Projects, overview, 27–30
  property sheet, 30–31
  reference element types, 13
  Reverse Engineering tool, overview, 47–51
  source document model element prefixes, 667
  Table Browser tool, overview, 52–53
  tools, overview, 20
  user interface control element types, 11–12
  Version Control tool, overview, 62–71
  X++ code editor, 31–33
MVC (Model-View-Controller), 494

N

Name, metadata property, 169
Named User license, 376–383
namespace, creating service contracts, 389–390
naming conventions
  assemblies, 74–75
  elements, 21–22
  label files, 35
  temporary tables, 584
navigation
  adding navigation items, overview, 182–183
  CreateNavigationPropertyMethods, 591–593
  Enterprise Portal and SharePoint integration, 235–236
  Enterprise Portal, web parts, 200
navigation bar, Enterprise Portal form design, 156–157
navigation layer forms
  design basics, 141–142
  Enterprise Portal design, 156–157
navigation pane, navigation layer forms, 141, 156–157
NetTcpBinding
  publishing services, 401
  system services, consuming, 404–407
NewModal, WindowMode, 221
NewWindow, WindowMode, 221
noFetch, 100
nonExists, 103
NonFatal, exception handling, 231
NotExistJoin, 164
NotInTTS cache, 446–452
NumberSeq
  getNextNumForRefParmId method, 425
Numeric, exception handling, 107

O

object creation operators, X++ expressions, 96
object models, UML, 49–50
object type
  overview, 91–92
  variable declaration syntax, 94
objects
  assemblies, calling managed code, 76
  RunOn properties, 422
observation, events, 521
OCC (optimistic concurrency control), 409
  Microsoft library resources, 577
  update_recordset, 434
OccEnabled, 100
OData feeds, SQL Server Power View reports, 335–340
ODC files, defining, 343
Office client integration applications, development of, 6–9
OLAP (online analytical processing) databases
  cubes, deploying, 303–309
  currency conversion logic, adding to cubes, 328–332
  DSV, overview of, 321
  Report Builder, 346
OnClick
  pop-up controls, 213–215
  validation, 231
OnlyFetchActive, metadata property, 170, 460
OpenPopup, 214–215
operating unit, defined, 634
operational persona, 334–335
operations resource framework
  extensions, 652–653
  MorphX model element prefixes, 654
  overview of, 648–652
  when to use, 652
optimistic concurrency control (OCC), 409
optimistic concurrency, performance, 444–446
optimisticlock
  performance and, 445–446
  select statements, 100
OptionalRecord, 168
organizational model framework
  custom operating units, creating, 639–640
  hierarchy designer, extending, 642
  integration with other frameworks application modules, 637–638
  organization hierarchies, 635–637
  organization types, 634–635
  overview, 634
  scenarios, modeling, 638–639
Origin, element IDs, 692
outer, 103
OuterJoin, joined data sources, 164
overrides, coding for, 184–186, 430

P

pack and unpack methods, 130–131, 202, 617
page definition element type, 15
Page Definition, Enterprise Portal web parts, 200
page rendering, Enterprise Portal architecture, 196–198
page templates, Enterprise Portal and SharePoint integration, 237–239
page title, Enterprise Portal web parts, 200
Page Viewer web part, 336–339
Page.IsValid, 231
pages, Enterprise Portal and SharePoint integration, 237–239
PageTitle, Enterprise Portal web parts, 200
parallel activities
  batch framework overview, 615
  workflow elements, 253
parameter methods, X++ syntax, 129
parentheses, X++ expressions, 96
parm methods, workflow classes, 269–270
Parse, formatting, 230
partitioned tables
  map element type, 10
  QueryFilter, 611–612
partitions
  architecture overview, 4–6
  cubes, deploying, 305–309
parts
  referencing from a form, 182
  types of, 181–182
Pascal casing, 93
PassClrObjectAcrossTiers, exception handling, 107
PastDataAccess, 363
patterns, X++ design and implementation, 128–133
pause statement, X++ syntax, 98
PBA, element prefix, 23
PDB (Program Database) files, third-party assemblies, 73–76
performance
  batch framework and, 466–467, 615
  caching

EntireTable cache, 453–454
overview, 446
record caching, 446–452
RecordViewCache, 454–455
SysGlobalObjectCache and SysGlobalCache, 456
unique index join cache, 452
client/server performance
reducing round-trips, 418–421
write tier-aware code, 422–426
coding patterns
execute X++ code as CIL, 466
firstfast, 476
indexing tips, 475–476
list page optimization, 476
loop iterations, reducing, 477–478
parallel execution, use of, 466–467
patterns for record existence checks, 472–473
repeat queries, avoiding, 473–474
SysOperation framework, 467–472
two queries vs. join, 474–475
compile and running X++ as .NET CIL, 126–128
configuration
AOS configuration, 463–464
client configuration, 464–465
extensive logging, 465
number sequence caching, 465
Server Configuration form, 463–464
SQL Administration form, 462
field lists, 456–462
InMemory temporary tables, 577–582
label file, 230
master scheduling and inventory closing, 465
Microsoft Dynamics AX services, overview, 415
monitoring tools
client access log, 490
database activities, monitoring, 488–489
Microsoft Dynamics AX Trace Parser, 479–488
Server Process ID (SPID), identifying, 489
Visual Studio Profiler, 490–491
overview, 417
restartable jobs and optimistic concurrency, 444–446
surrogate keys, 586
table inheritance and, 598–599
transaction performance
delete_from operator, 435–436
insert_recordset operator, 429–432
overview, 426–427
RecordInsertList and RecordSortedList, 437–439
set-based data operators, overview, 427–428
set-based operations and table hierarchies, 428
transferring code into set-based operations, 439–444
update_recordset operator, 432–435
Unit of Work, 599–601
PerInvocation, extensible data security, 367
period templates, time period filters, 344–345
Periodic, Area Page design, 145
permissions
debugging security, 373–375
form controls, 359
forms, 356–359
privileges, creating, 359–361
security framework overview, 353–356
table permissions, coding, 369–371
PerSession, extensible data security, 367
personas, defined, 334–335
perspective element type, defined, 11
perspectives, metadata, 325–326
pessimisticLock, select statements, 100
pessimisticlock, update_recordset, 434
PivotTables, displaying in Role Centers, 340
placeholders, labels, 36–37
policies
debugging security, 373–375
extensible data security policies, creating, 364–369
extensible data security policies, debugging, 368–369
organization model framework, 637
policy context, defined, 365
policy query, defined, 365
queries, data security policies, 365–369
security framework overview, 353–356
policy modeling, 5
polymorphic associations, 91, 132, 596–598
pop-up controls, AxPopup, 213–215
Popup, window type, 174
PopupClosed, 213–215
ports, publishing services, 401
post event handlers, 522–523
postback
updates, 204
validation, 231

PostBackTrigger, 222
postBuild, 494
postRun, 494
Power View, SQL Server
  reports, displaying, 335–340
  reports, editing, 339–340
PowerPivot, 335–340
pre-event handlers, 522–523
prefix, element naming conventions, 21–22
PreRender, chart reporting override, 296
presentation tier
  architecture, 8–9
  Business Intelligence (BI), 299–300
primary entities, details form, 151
primary table, 365–369
PrimaryIndex
  record caching, 447
  surrogate keys, 586–587
print statement, X++ syntax, 98
Private and Shared Projects, 27
privilege element type, 14
privileges
  creating, 359–361
  security framework overview, 353–356
process cycle element type, 14
process cycles, defined, 355
Process State
  accounting, 662–663
  source document framework, 664–665
processingMode, 412
procurement and sourcing, organization model framework, 638
Prod, element prefix, 23
Producer, eventing, 520
product builder, element prefix, 23
Product Configuration, 646–647
Product Dimensions, 643
Product Master, 643, 646–647
product model framework
  extension of, 647–648
  overview, 643–647
  when to use, 647
Product Variant, 643, 646–647
production, element prefix, 23
Program Database (PDB) files, third-party assemblies, 73–76
project, element prefix, 23, 23
Projects
  authoring managed code, 77–84
  automatic generation of, 27–29
  creating, 27
  development tools for, 29
  project types, 30
  upgrades, Compare tool, 56
Projects tool, 20
property sheet, 30–31
Property sheet tool, 20
providers, workflows, 254–255, 260–261
ProviderView, 203
proxies. *See also* .NET Framework
  authoring managed code, 77–84
  Enterprise Portal, developing for, 226–228, 231–232
  Enterprise Portal, security, 232–235
  working with, overview, 73
public key, assembly names, 74–75
PublicPage, 239
publisher, Help system, 550
publishing services, 400–401
publishing, models, 694–699
Purch, element prefix, 23
Purchase Order form, designing, 153–155
purchase, element prefix, 23
purpose, organizational hierarchy, 636

Q

quality checks, Version control tool, 65
queries
  Axd queries, creating, 395–396
  AxFilter, 209–210
  cues, Role Center page design, 143
  data-aware statements, X++ syntax, 99–104
  dimension framework query data, 658–659
  Enterprise Search, 240–243
  field lists, performance and, 456–462
  form queries, overview, 170–172
  full-text support, 606–607
  Microsoft Office client integration, 190
  performance, indexing tips, 475–476
  policy queries, data security, 365–369
  polymorphic, table inheritance and, 165–167
  query element type, 11
  Query object, form queries, 172
  Query service, Enterprise Search, 240–243
  QueryBuildDataSource, 170–172
  QueryBuildRange, 172
  QueryFilter, 172, 607–612
  QueryRun object, form queries, 172

QueryRunQueryBuildDataSource, 171
repeat queries, avoiding, 473–474
system services, 388, 404–407
table inheritance, 594–599
table relations, 588–593
timeout, 463
two queries vs. join, 474–475
workflow documents, 250
Workflow wizard, 251–252
queues
workflow message queue, 257
workflows, overview, 253–254
Quick links, Enterprise Portal, web parts, 200
QuickLaunch, Enterprise Portal web parts, 200
QuickLaunchDataSource, 235
QuickLink, Role Center, 143
quid
value types, overview, 88
variable declaration syntax, 94

R

RDCE (Microsoft Dynamics AX Report Definition Customization Extension), 286–288
RDL (report design definition), 287–288
read permissions
forms, 356–359
menu items, 360
ReadPermissions, 360
real value types, overview, 88
real variable declaration syntax, 94
RecId, table inheritance storage model, 596
record buffers
InMemory temporary tables, 578–582
run-time temporary tables, 585
record caching, 446–452
record context, Enterprise Portal security, 235
record types
reference types, overview, 89
variable declaration syntax, 94
RecordInsertList, 437–439
record-level security (RLS), 430
RecordSortedList, 437–439
RecordViewCache, 454–455
reference types, 13, 89
References, third-party DLLs, 75–76
reflection
classIdGet system function, 672–673
common type, 91
dictionary API, 676–680
instrinsic functions, 670–671
overview, 669–670
table data API, 673–676
treenodes API, 680–685
typeOf system function, 671–672
RefRecId, 132–133
refresh
cacheAddMethod, 419
elements, 25
RefreshFrequency, 367
RefTableId, 133
regulatory requirements, segregation of duties, 354
RelatedTableRole, 593
relational operators, X++ expressions, 96
relations, table, 588
relationships, entity relationship data model, 51
RELATIONTYPE, 596
Released Product, 645
reliable asynchronous mode, SysOperations, 468
Remote Procedure Call (RPC)
grouping calls, 425
MorphX development environment, 6–7
passing table buffers, 425–426
repeatableRead, 100
reporting. *See also* Role Center pages
AxReportViewer, 216
client-side solutions, 276–277
data processing extensions, 288
default chart format, override, 296
Enterprise Portal charts
binding chart control to dataset, 292
chart development tools, 289
data series binding, 292–294
EP Chart Control, creating, 290–291
mark-up elements, 291–292
overview, 289
execution sequence, overview, 278–279
interactive functions, adding, 294–295
list pages as alternatives, 148–149
Microsoft Dynamics AX extensions, 286–288
model elements for reports, 282–285
overview, 275–276
planning for, 279–281
production reports, overview, 281–282
rendering, MorphX user interface control element types, 11–12
Report Builder reports, 346

reporting *(continued)*
Report Definition Language (RDL), 282
Report Deployment Settings form, 288
report design definition (RDL), 287–288
report element type, MorphX, 12
Report, Enterprise Portal web part, 200
Reporting Services, data connection file (.rsds), 336
Reports, Area Page design, 145
requirements, identifying, 324–325
server-side solutions, 277–278
SQL Server Power View, displaying data, 335–340
SQL Server Reporting Services (SSRS), 3–6, 8
SSRS extensions, 285
troubleshooting, 296–297
Visual Studio tools, 346–349
requirements, element prefix, 23, 23
Requirements, operations resource framework, 650–651
ResetFilter, 209
resource element types, 16, 22
Resource Group, operations resource framework, 648–652
resource modeling, processing architecture, 5
Resource, operations resource framework, 648–652
REST API, Excel report display, 340
restartable jobs, performance, 444–446
restore, refreshing elements, 25
retail channel, defined, 635
retries, batch jobs, 628
retry statement, X++ syntax, 98
return statement, X++ syntax, 98
Reverse Engineering tool, 20, 47–51, 669
reverse, data-aware statements, 100
Role Center pages
designing, 142–144
displaying analytic content
Business Overview and KPI List web parts, 341–345
Excel reports, 340
overview, 333–335
presentation tools, choice of, 335
Report Builder, 346
SQL Server Power View, 335–340
Visual Studio tools, 346–349
Quick links, 200
user profiles, 308–309
role element type, 14
role-based access control, 353–356
role-based security element types, 14
RoleName, 367
RoleProperty, 367
roles, security
assigning privileges and duties, 361–362
assigning to users, 363–364
customization and licensing, 383
debugging, 373–375
security framework overview, 353–356
role-tailored design, overview, 139–140
root data sources, 164–169
root table, creating, 594
RootTableContext, 223–225
Row, AxGridView, 208
RowCount, 104
RPC (Remote Procedure Call)
grouping calls, 425
MorphX development environment, 6–7
passing table buffers, 425–426
rules, Best Practices tool, 40–43
run method
batch class, creating, 616
datasets, Enterprise Portal, 202
RunAsPermission, 371–372
RunBase. *See also* SysOperations
client/server considerations, 516
inheritance, 510–511
overview, 510
pack-unpack pattern, 512–516
performance and, 420–421
property method pattern, 511–512
SysOperation comparison, 495–510, 613
UML object model, 49–50
RunBaseBatch, 495–496, 616–617
runBuf, code access security, 124–126
RunOn, object instantiation, 422
run-time
control modifications, 174
QueryFilter API, 611–612
table relations, 588–593
workflow, 257–260

S

sales and marketing management, element prefix, 23
Sales Order form, designing, 153–155
Sales, element prefix, 23

Sarbanes-Oxley, segregation of duties, 354
Save button, autogeneration, 220
saveData, 653
SaveDataPerPartition, 611
scheduled batch mode, SysOperations, 468
scheduling, batch framework, 614
schema
    AxdDocument class, 393–394
    prebuilt cubes, analytic data, 310
    reusing tables and fields, creating cubes, 333
    XML schema definitions for message envelopes, 410–411
Script icon, X++ Code Editor, 33
scripts
    execute editor script, 32
    running, shortcut keys X++ code editor, 32
    X++ Code Editor, overview, 33
search
    Enterprise Portal, navigation form design, 156–157
    Find tool, overview, 53–54
    incremental, shortcut key, 32
    search bar, navigation layer forms, 141
    Search Configuration Wizard, 241–243
Secure Sockets Layer (SSL), Enterprise Portal, 232–235
security
    assembly names, 74–75
    best practices, overview, 372–373
    code access security (CAS), 124–126, 371–372
    configuration key element types, 16–17
    debugging, 373–375
    Enterprise Portal, developing for, 232–235
    extensible data security policies, creating, 364–369
    hiding report columns, 286–288
    insert_recordset operator, 430
    license code element types, 16–17
    permissions, controls, 359
    permissions, forms, 356–359
    permissions, server methods, 359
    privileges, assigning to security roles, 361–362
    privileges, creating, 359–361
    role-based security element types, 14
    RunBase, 420
    security artifacts, developing, 356–363
    security framework overview, 351–356
    security role assignments, 355
    service operations, overview, 400
    table permissions, coding, 369–371
    valid time state tables, use of, 362–363
    validate security artifacts, 363–364
security policy element type, 14
security roles
    assigning privileges and duties, 361–362
    assigning to users, 363–364
    customization and licensing, 383
    debugging security, 373–375
segregation of duties, 354
select
    data-aware statements, X++ syntax, 99–104
    field lists, performance and, 460
    select query, sample code, 101
select forUpdate, 104
SelectedIndexChanged, 208
selection, shortcut keys, 32
selectionChanged, SysListPageInteractionBase, 218
semicolon, use in X++, 95
Send API, 411–414
Sequence, exception handling, 107
serialization, pack and unpack methods, 130–131
Server Configuration form
    batch server, configuring, 625–626
    performance and, 463–464
Server Configuration Utility, hot-swapping assemblies, 84–85
server license, 376–383
server methods, permissions for, 359
Server Process ID (SPID), 489
server, method modifier, 119
service contracts, creating, 389–390, 392–393
service group element type, 13
service implementation class, 388–391
service management, element prefix, 23
service-oriented architecture, 386
Service References, 387
services. *See* Microsoft Dynamics AX services
services element types, 13
session caching, Enterprise Portal, 223
session disposal, Enterprise Portal, 223
SetAsChanged, AxFilter, 209
set-based data operators
    delete_from operator, 435–436
    InMemory temporary tables, 578
    overview, 427–428
    table hierarchies and, 428
    tranferring code into, tips for, 439–444
    update_recordset operator, 432–435

setButtonEnabled, SysListPageInteractionBase, 218
setButtonVisibility, SysListPageInteractionBase, 219
SetFieldValue, AxPopupField, 214–215
setGridFieldVisibility, SysListPageInteractionBase, 219
SetMenuItemProperties, AxToolbar, 212
setter method, 592
setTmp, run-time temporary tables, 585
setTmpData, 579–582
Setup, Area Page design, 145
Shared Projects, overview of, 27
shared steps, 536–539
SharePoint
  Enterprise Portal, developing for, 216–217
  Enterprise Portal, integration with
    Enterprise Search, 240–243
    site definitions, page templates, and web parts, 237–239
    site navigation, 235–236
    themes, 243
    web part page, import and deploy, 239–240
  KPIs, adding to KPI List web part, 342–344
  reporting, troubleshooting, 297
SharePoint Server
  architecture, 8
  Enterprise Portal architecture and, 196–198
  Power View reports, displaying, 336–339
  processing architecture, 3–7
SharePoint Services server, 336–340
SharePoint web client applications, development of, 6–9
shift operators, X++ expressions, 96
shop floor controls, element prefix, 23
shortcut keys
  debugger tool, 47
  X++ code editor, 32
ShowContextMenu, AxGridView, 208
ShowExpansion, AxGridView, 208
ShowFilter, AxGridView, 208
ShowLink, 236
signatures, digital
  assembly names, 74–75
  models, signing, 696–697
SimpleList template, 162
SimpleListDetails template, 162
site definitions, Enterprise Portal and SharePoint integration, 237–239
skipAosValidation, 431–432, 434, 436
SkipAOSValidationPermission, 371–372
skipDatabaseLog, 431–432, 434, 436
skipDataMethods, 431–432, 434, 436
skipDeleteMethod, 436
skipEvents, 431–432, 434, 436
SMA, element prefix, 23
SMM,element prefix, 23
source code files, 688
Source Code, Titlecase Update tool, 93
source document framework
  MorphX model element prefixes, 667
  overview, 664–665
  when to use, 665
sourcing and procurement, organization model framework, 638
specialized base enumerations, value types, 88
specialized primitive types, value types, 88
SPID (Server Process ID), 489
SPLinkButton, 212
SQL Administration form, performance and, 462
SQL Server
  architecture, 7
  processing architecture, 3–6
SQL Server Analysis Services (SSAS)
  prebuilt BI solutions, implementing, 301–309
  prebuilt projects, modifying, 319–323
  processing architecture, 3–8
  SSAS server, configuring, 302
SQL Server Analysis Services Project Wizard
  cubes, customizing, 311–319
  cubes, generating and deploying, 303–309, 328–333
  currency conversions, 316–317
  deploying projects, 301
SQL Server Power View
  reports, displaying, 335–340
  reports, editing, 339–340
SQL Server Reporting Services (SSRS)
  architecture, 3–8
  AxReportViewer, 216
  client-side reporting solutions, 276–277
  data processing extensions, 288
  default chart format, override, 296
  Enterprise Portal, web parts, 200
  interactive functions, adding, 294–295
  Microsoft Dynamics AX extensions, 286–288
  model elements for reports, 282–285
  processing architecture, 3–6, 8
  production reports, overview, 281–282
  report execution sequence, overview, 278–279
  report solutions, planning for, 279–281
  reporting, overview, 275–276
  server-side reporting solutions, 277–278

SSRS extensions, creating, 285
troubleshooting, reporting framework, 296–297
SqlDataDictionaryPermission, 372
SqlStatementExecutePermission, 372
SSAS (SQL Server Analysis Services)
prebuilt BI solutions, implementing, 301–309
prebuilt BI solutions, modifying, 319–323
SSAS server, configuring, 302
SSL (Secure Sockets Layer), Enterprise Portal security, 232–235
SSRS report element type, MorphX, 12
Standard, TabPage controls, 176–178
Standard, window type, 174
startOperation, 494
Startup Element, debugging managed code, 81–82
state model, workflows, 266–267
StateManager, 266–267
statements, X++ syntax, 96, 99–104. *See also* X++ programming language (code)
static CLR elements, invoking, 109
static file element type, 15
Static Files, Enterprise Portal and SharePoint integration, 237
static RDL reports, 288
Static Report Design, 288
static, method modifier, 119
status bar, navigation layer forms, 142
Storage Dimension Group, 645
str
value types, overview, 88
variable declaration syntax, 94
strategic persona, 334–335
strFmt, 36–37
string concatentation, X++ expressions, 96
style sheet themes, Enterprise Portal and SharePoint integration, 243
Subledger Journal, 660–662
SubMenu, AxToolbar, 212
SubmitToWorkflow, action menu items, 272
SubmitToWorkflow, workflow artifacts, 265
subworkflows, workflow elements, 253
sum, X++ select statements, 102
summary page, Help system, 550–551
SupportInheritance, 594
surrogate foreign keys
CreateNavigationPropertyMethods, 591–593
performance and, 168, 586–587
table relations, 590–591
surrogate keys, overview of, 585–587
SvcConfigUtil, publishing services, 401
switch statement, X++ syntax, 98
synchronization
AxFilter, 209
elements, refreshing, 25
proxies, 82–84
temporary tables, 584
Version control tool, 67–68
synchronization log, viewing, 68
synchronous mode, SysOperations, 468
Sys, element prefix, 23
SysAnyType, 90
SysBPCheck, 42–43
SysBPCheckMemberFunction, 42–43
SysClientAccessLog, 490
SysDatabaseLogPermission, 372
SysEntryPointAttribute, 359, 371, 400
SysGlobalCache, performance and, 456
SysGlobalObjectCache, performance and, 456
SysListPageInteractionBase, 218–219
SysModel, reflection table, 675
SysModelElement, reflection table, 675
SysModelElementData, reflection table, 675
SysModelElementLabel, reflection table, 675
SysModelElementSource, reflection table, 675
SysModelElementType, reflection table, 675
SysModelLayer, reflection table, 676
SysModelManifest, reflection table, 676
SysModelManifestCategory, reflection table, 676
SysOperation
attributes, 495
classes, 494
creating batch-executable class, 616–617
overview, 493–494
RunBase comparision, 495–510
SysOperationAutomaticUIBuilder, 494
SysOperationContractProcessingAttribute, 495
SysOperationController, 495–496
SysOperationDisplayOrderAttribute, 495
SysOperationHelpTextAttribute, 495
SysOperationLabelAttribute, 495
SysOperations
overview, 420, 613
performance, 467–472
SysOperationUIBuilder, 494
SysPackable interface, 495–496
SysQueryForm, timeout settings, 463
SysTableBrowser, 52–53
system documentation element type, 16

system function statement, X++ syntax, 98
system services
consuming, 404–407
overview, 387–388
system workflows, defined, 246–249. *See also* workflow
SystemAdministratorHelpOnTheWeb, 549
SystemFatal, exception handling, 232
SystemFilter, 209
SystemManaged, form permissions, 358
SystemTable, 611
SysTest framework, new features, 527–533
SysTestCheckInTestAttribute, 528–533
SysTestFilterStrategyType, 531–533
SysTestInactiveTestAttribute, 528–533
SysTestListenerTRX, 541
SysTestMethodAttributes, 528
SysTestNonCheckInTestAttribute, 528–533
SysTestTargetAttribute, 528
SysVersionControlFileBasedBackEnd interface, 71

T

table. *See also* temporary tables
buffers, passing by value, 425–426
buffers, set-based data operators, 428
document services, customizing, 397
inheritance, overview, 165–167
permissions, coding, 369–371
reference types, overview, 89
table data, reflection API, 669–670, 673–676
table hierarchies and set-based operators, 428
table index, database query sample, 101
table maps, common type, 91
table relations, overview, 588
table-level patterns, 131–133
Table Browser tool, 20, 52–53
table collection element type, defined, 11
table element type, defined, 10
table inheritance
modeling, 594–595
performance and, 598–599
polymorphic behavior, 596–598
storage model, 596
table of contents, Help system, 550, 563–565
Table References, 589
Table, metadata property, 169
TableContextList, Enterprise Portal, 223–225
TableDataKeys, Enterprise Portal, 224–225
TableOfContents template, 162
TabPage control, 176–178
tabular models, PowerPivot, 336–340
tactical persona, 334–335
TargetClassTest, 528
TargetControl, AxPopup, 215
TargetControlID, AxLookup, 211
TargetId, AxPopupField, 214–215
tasks
code compiler, 37–39
logical approval and task workflows, 260–262
task modeling, performance and, 466–467
workflow artifacts, 265
workflow elements, 252
Tax, element prefix, 23
Team Foundation Build, 540–543
team, defined, 635
TempDB
creating temporary tables, 583–585
extensible data security constructs, 367
overview, 582–583
performance and, 423–424
transferring code into set-based operations, 442–444
templates
Dynamics AX Reporting Project, 282
Enterprise Portal and SharePoint integration, 237–239
EP Chart Control, 290–291
Excel, 190–191
form patterns, 160–162
Help content, 551–552
time period filters, 344–345
user templates, adding, 192–193
Word, 191–192
workflow types, 251–252
temporary tables
creating, 583–585
EntireTable cache, 453–454
inMemory, 422–423, 428
InMemory, 578–582
insert_recordset operator, 429–432
TempDB, overview, 582–583
TempDB, performance and, 423–424
transferring code into set-based operations, 442–444
Terminal Services, performance, 464
Test Listeners, 541
Test Manager. *See* Microsoft Test Manager

testing
new features, 527–533
test selection, 542–544
Visual Studio tools
acceptance test driven development (ATDD), 535–536
ALM solution, 534
ordered test suites, 539–540
overview, 533–534
shared steps, 536–539
Team Foundation Build, 540–543
test case evolution, 538
testsElementName, 528
TextBox, AxPopupField, 214–215
TFS (Visual Studio Team Foundation Server), 62–64
themes, Enterprise Portal and SharePoint integration, 243
third-party assemblies, .NET Framework and, 73–76
third-party integration applications
development of, 6–9
presentation tier architecture, 9
throw statement
exception handling, 105–106
X++ syntax, 98
time
date-effective framework, 601–606
time period filters, Business Overview web part, 344–345
TimeOfDay, value types, 88
TimeOfDay, variable declaration syntax, 94
valid time state tables, 355, 362–363
Timeout
exception handling, 107
Server Configuration form, 463
title, Enterprise Portal web parts, 200
Titlecase Update tool, 93
TitleDateSource, form properties, 174
TODO tasks, code compiler, 37–39
Toolbar, Enterprise Portal web part, 200, 212–213
topics, Help system
add labels, fields, and menu items, 559–561
context-sensitive topics, 561–562
create in HTML, 552–559
non-HTML topics, creating, 565–567
overview, 549–550
TopNavigationDataSource, 235
Trace Parser, 479–488
Tracking Dimension Group, 645
transaction details form, designing, 153–155
transaction performance
delete_from operator, 435–436
overview, 426–427
set-based data operators, overview, 427–428
update_recordset operator, 432–435
transaction tracking system (TTS), grouping calls, 425
Transact-SQL
tracing statements, 488–489
transferring code into set-based operations, 442–444
TransDate, currency conversion logic, 330–332
Translations, customizing cubes, 315–316
travel and expense, organization model framework, 638
treenodes, reflection API, 670, 680–685
TreeNodeType, 683–685
troubleshooting
Help system, 572–573
reporting framework, 296–297
tracing code, 487–488
Trustworthy Computing, 372–373
try statement, X++ syntax, 98
TTS (transaction tracking system), grouping calls, 425
ttsAbort, 104
InMemory temporary tables, 581–582
ttsBegin, 104
InMemory temporary tables, 581
ttsCommit, 104
InMemory temporary tables, 581
Tutorial_CompareContextProvider, 58–59
Type Hierarchy Browser tool
overview, 20
table inheritance hierarchy, 594–595
type hierarchies, 89–93
Type Hierarchy Context tool, 20, 89–93
type hierarchy, Cross-reference tool, 60
typed data source, element prefix, 22
typeOf system function, reflection, 669, 671–672

U

Unified Modeling Language (UML)
object models, 49–50
Reverse Engineering tool, overview, 47–51
Unified worklist web part, 201, 256
unique index join cache, 452
Unit of Work
form data overview, 167–168
overview of, 599–601

unpack method, 130–131, 203, 617
update method, field lists, 458
update permissions
  forms, 356–359
  menu items, 360
update_recordset
  table hierarchies and, 428
  transaction performance, 104–105, 432–435
  transferring code into set-based operations, 442–444
UpdateConflict, 107
UpdateConflictException, 107
UpdateConflictNotRecovered, 107
UpdateOnPostback, 204
UpdatePanel
  AxContentPanel, 215
  Enterprise Portal, AJAX, 222
UpdatePermissions, 360
updates
  business documents, consuming services, 407–409
  date-effective framework, 605–606
  Help content, 562–563
  User control web part, 201
upgrades
  Compare tool, 56
  Project and, 29
URL
  Excel reports, displaying, 340
  Help system, AOS, 549
  Power View reports, displaying, 336–339
URL web menu item, 218
U.S. Food and Drug Administration regulations, 354
UseIntList, 391
user access
  security framework overview, 351–356
  validate security artifacts, 363–364
user client access license (CAL), overview, 376–383
User control web part
  Enterprise Portal architecture, 196–197
  Enterprise Portal, AJAX, 222
  Enterprise Portal, overview, 201
user experience, designing
  area pages, 144–146
  details form, 150–153
  Enterprise Portal web client experience, 155–157
  list pages, 146–150
  navigation layer forms, 141–142
  overview, 137–139
  Role Center pages, 142–144
  role-tailored design, 139–140
  transaction details forms, 153–155
  user feedback, importance of, 157–158
  work layer forms, 142
user interface
  control element types, MorphX, 11–12
  Enterprise Portal architecture, 196–198
  labels, localization of, 33
  SysOperationUIBuilder, 494
user profiles, deploying cubes, 308–309
user session info service, 388
user-defined class types, 89
UserDocumentation, Help system, 549, 571
UserFilter, 209
users
  creating, 363
  roles, assigning, 363–364
utcDateTime
  relational modeling, 602–603
  value types, overview, 88
  variable declaration syntax, 94
UtilElements, reflection table, 676
UtilIdElements, reflection table, 676
UtilModels, reflection table, 676

V

valdiation
  Validate property, associations, 48–49
valid time state tables, security, 362–363
validation
  aosValidateDelete, 436
  aosValidateInsert, 438–439
  aosValidateRead, 433
  aosValidateUpdate, 433
  AxdDocument class, 394
  cross-table, document services, 393
  document services, customizing, 397–399
  Enterprise Portal, developing for, 231
  registering methods, postRun, 494
  report server validation, troubleshooting, 297
  skipAosValidation, 431–432, 434, 436
  Table Browser tool, 52–53
  table permissions coding, 370–371
  validateByTree, 656
ValidFrom
  date-effective tables, consistency, 604–606

valid time state tables, 362–363
ValidTimeStateFieldType, 601–603
validTimeState, 100
validtimestate key, 602–603
ValidTimeStateFieldType, 362–363, 601–603
ValidTimeStateMode, 604–606
ValidTo, 362–363, 601–606
value stream, defined, 635
value types, 88
ValueFormatter, Enterprise Portal, 230
values, X++ expressions, 95
variables
Auto variables, overview, 187
common type, 91
declarations, X++ syntax, 93–95
expressions, X++ syntax, 95
extended data type, 92–93
object type, 91–92
reference types, overview, 89
value types, overview, 88
X++ syntax, camel casing, 93
variant configuration technology, Product Master, 646–647
Vend, element prefix, 23
vendors, element prefix, 23
Version Control tool
common tasks, 65
create a build, 71
element history, 69
element life cycle, 64–65
integrating with other version control systems, 71
overview, 20, 62–64
pending elements, viewing, 70
revisions, comparing, 70
vertical application domain partition, 4–6
VerticalTabs, TabPage controls, 176–178
view element type, defined, 10
view type, reference types, 89
ViewState, Enterprise Portal, 228–229
ViewUserLicense, 383
Visio, Reverse Engineering tool, 47–51
Visual SourceSafe (VSS), 62–64
Visual Studio
analytic reports, tools for, 346–349
authoring managed code, 77–84
batch jobs, debugging, 630–631
details page, creating, 219–221
Enterprise Portal architecture, 196–198
Enterprise Portal, developing for, 216
model elements for reports, 282–285
prebuilt projects, modifying, 319–323
presentation tier architecture, 8–9
proxies, Enterprise Portal, 226–228
report execution sequence, overview, 278–279
reporting
chart controls, 291–292
client-side solutions, 276–277
data processing extensions, 288
default chart format, override, 296
interactive functions, adding, 294–295
Microsoft Dynamics AX extensions, 286–288
overview, 275–276
production reports, overview, 281–282
report solutions, planning for, 279–281
server-side solutions, 277–278
SSRS extensions, creating, 285
troubleshooting, reporting framework, 296–297
test tools
acceptance test driven development (ATDD), 535–536
ALM solution, 534
ordered test suites, 539–540
overview, 533–534
shared steps, 536–539
Team Foundation Build, 540–543
test case evolution, 538
test selection, 542–544
third-party assemblies, using, 73–76
Visual Studio Integrated Development Environment (IDE)
overview, 7
processing architecture, 4–6
Visual Studio Profiler, performance monitoring, 490–491
Visual Studio Team Foundation Server (TFS), 62–64
VSS (Visual SourceSafe), 62–64

W

warehouse management, element prefix, 23
warehouses, external data integration, 322–323
warnings, compiler
overview, 37–39
suppressing, Best Practices tool, 41–42
WCF (Windows Communication Foundation), 7–8
web client element types, overview, 14–15

web content element type, 15
web control element type, 15
web frameworks, element prefix, 23
web menu
  AxActionPanel, 211–212
  web menu element type, 14
  web menu item type, 14
  workflow menu items, 252
web module element type, 15
web part element type, 15
web part page
  Enterprise Portal and SharePoint integration, 239–240
  Enterprise Portal architecture, 196–197
web parts
  Enterprise Portal and SharePoint integration, 237–239
  Enterprise Portal, overview, 199–201
  Enterprise Portal, security, 233–235
web platforms. *See* Enterprise Portal
web services
  external, consuming, 414
  Microsoft Internet Information Services, 8
Web, element prefix, 23
web.config
  publisher, adding, 569–570
  session disposal and caching, 223
WebLink, SharePoint site navigation, 236
WebMenuName, 211–212
WF (Windows Workflow Foundation), 249–250
while statement, 99, 103
Window Performance Monitor, 482–483
window statement, X++ syntax, 99
WindowMode, 220–221
Windows client applications
  development of, 6–9
  presentation tier architecture, 8
Windows Communication Foundation (WCF), 7–8
Windows Search Service, Help system, 549
Windows Server AppFabric, 223
Windows Workflow Foundation (WF), 249–250
WindowSize, 220–221
WindowType, 174
Wizard wizard, 29
wizards
  Create New Document Service Wizard, 393
  Label Files wizard, 35
  Project development tools, 29
  Search Configuration Wizard, 241–243
  SQL Server Analysis Services Project Wizard, 301, 303–309
  Wizard wizard, 29
  Workflow wizard, 251–252
WKEY, Enterprise Portal security, 235
WMS, element prefix, 23
Word
  architecture, 9
  templates, building, 191–193
work layer forms
  Enterprise Portal design, 157
  overview of, 142
workflow
  activation of, 270–274
  architecture, 256–262
  categories, creating, 268
  creating artifacts, and business logic, 264–265
  display menu item, adding, 270
  document class, creating, 268–270
  elements of, 252–253
  event handlers, 252
  implementation, overview, 263–264
  infrastructure for, 246–249
  logical approval and task workflows, 260–262
  menu items, 252
  overview, 245–246
  providers, 254–255
  queues, 253–254
  state management, 266–267
  Windows Workflow Foundation (WF), overview, 249–250
  work items, 256
  workflow categories, 251
  workflow document and workflow document class, 250
  workflow editor, 255–256
  workflow instances, 256
  workflow life cycle, 262–263
  workflow run time, 257–260
  workflow types, 251–252
Workflow
  activatefromWorkflowConfiguration, 273
  activatefromWorkflowSequenceNumber, 273
  activatefromWorkflowType, 273
workflow approval element type, MorphX, 12
workflow category, workflow artifact, 264
workflow document class, workflow artifacts, 265
workflow document query, workflow artifacts, 265
workflow element types, MorphX, 12–13

workflow provider element type, MorphX, 13
workflow started message, 259–260
workflow task element type, MorphX, 12
workflow type element type, MorphX, 12
workflow type, workflow artifacts, 265
Workflow wizard, 251–252
WorkflowDataSource, 271
WorkflowDocument, 269
WorkflowEnabled, 271
WorkflowType, 271
Workspace, window type, 174
WREC, Enterprise Portal security, 235
write method, 168
write tier-aware code, performance and, 422–426
writing method, 168
written method, 168
WSS (Windows Search Service), 549
WTID, Enterprise Portal security, 235

X

X++ code editor tool
  overview, 20, 31–33
  shortcut keys, 32
X++ collections, data contracts, 391
X++ programming language (code). *See also* MorphX
  assemblies, calling managed code, 76
  attributes, 123–124
  batch jobs, debugging, 630–631
  classes and interfaces, overview, 117–118
  CLR interoperability, 108–112
  code access security (CAS), 124–126
  code element types, 13
  COM interoperability, 112
  comments, syntax for, 115
  compiler, overview, 37–39
  compiling and running X++ as .NET CIL, 126–128
  data-aware statements, 99–104
  date-effective tables, data retrieval, 603–604
  debugger tool, overview, 43–47
  delete_from operator, 435–436
  delgates, 120–122
  design and implementation patterns, 128–133
  dictionary, reflection API, 676–680
  exception handling, 105–108
  executing as CIL, 466
  expressions, 95
  fields, 118
  instrinsic function, reflection, 670–671
  introduction to, 87
  Jobs, 88
  macros, 113–115
  methods, 118–120
  MorphX development environment, 6–7
  pre- and post-event handlers, 122–123
  processing architecture, 5
  proxies, Enterprise Portal, 226–228
  RecordSortedList and RecordInsertList classes, 438–439
  reference types, 89
  referencing labels from, 36–37
  reflection, overview, 669–670
  set-based data operators, overview, 427–428
  set-based data operators, transferring code into, 439–444
  statements, overview, 96
  syntax, overview, 93
  table data, reflection API, 673–676
  table element type, 10
  treenodes, reflection API, 680–685
  troubleshooting, tracing, 487–488
  type hierarchies, 89–93
  type system, 88–93
  update_recordset operator, 432–435
  value types, 88
  variable declarations, 93–95
  XML documentation, 116
xClassTrace, 483
XDS (extensible data security framework)
  debugging data security policies, 368–369
  organization model framework integration, 356
  policies, creating, 364–369
  temporary table constructs, 368
XML documentation
  AxdDocument class, 393–394
  EPSetupParams, 237
  header insert, shortcut key, 32
  Help system, table of contents, 563–565
  schema definitions for message envelopes, 410–411
  system services, consuming, 406–407
  third-party assemblies, using, 73–76
  X++ syntax, 116
XMLHttpRequest, 222
XPO files, 688
Xpp-PrePostArgs, 122–123
xRecord type, reference types, overview, 89

About the authors

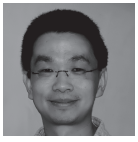
Principal authors

Anees Ansari is a program manager in the Microsoft Dynamics AX product group. His areas of focus include Enterprise Portal and web-based frameworks and clients for Microsoft Dynamics AX. He is passionate about web-based technologies and has been working in that area for more than 11 years, including 7 years at Microsoft.

Anees has a broad range of experience in various roles, both within and outside Microsoft. His last role was technical product manager in the Microsoft Web Platform and Standards group, where he was responsible for product management and marketing strategy for Microsoft ASP.NET and Microsoft Visual Web Developer products. Prior to that, he was a software developer on the Outlook Web App team working on Microsoft Exchange Online and Microsoft Exchange Server products. Before joining Microsoft, Anees worked with start-ups that helped small to mid-sized companies increase their online business and customer base by designing, developing, and managing their web portals.

Anees has a master's degree in computer science from the University of South Florida and a certificate in business fundamentals from the Kelley School of Business at Indiana University.

David Chell is a senior technical writer on the Service Industries and Retail Content Publishing team for Microsoft Dynamics AX.



Zhonghua Chu is a principal development lead on the Microsoft Dynamics AX server team. He has worked on data access and other server-related areas since Microsoft Dynamics AX 4.0. Zhonghua joined the Microsoft SQL Server Data Warehouse team after graduating from the University of Wisconsin–Madison, and has been working on application system design and implementation for over 13 years.

Dave Froslie is a principal test architect on the Microsoft Dynamics AX development team. He joined Microsoft in 2002, and has held a variety of development and test leadership roles in business solutions and development tools. Before joining Microsoft, Dave was a development manager for teams building engineering test systems at MTS Systems in Eden Prairie, Minnesota.

In his current role, Dave is responsible for providing guidance on test approaches for the product, with a focus on automated tools, libraries, and infrastructure. Dave also has a strong interest in engineering processes, particularly agile development. Blog posts that Dave has written on these and other topics can be found at *http://blogs.msdn.com/b/dave_froslie/*. Dave works in the Microsoft development office in Fargo, North Dakota, and lives across the river in Minnesota with his wife, Dawn, and daughter, Ali.

Chris Garty is a senior program manager on the Microsoft Dynamics AX Client Presentation team in Fargo, North Dakota. Chris joined the Microsoft Dynamics AX team during the Microsoft Dynamics AX 2009 development cycle. During the Microsoft Dynamics AX 2012 development cycle, Chris helped guide the changes to List Pages and Details forms, and worked on a range of user experience components. Chris's role on the Microsoft Dynamics AX team has recently expanded to cover integration with Microsoft Office and document management.

Chris has 13 years of experience in software development and consulting, the last 8 of which have been spent at Microsoft.

Chris was born and raised in New Zealand, and he is lucky enough to visit New Zealand and Australia almost yearly to see his family. He moved to Fargo to work for the best company in the world and lives there, eight winters and a couple of floods later, with his wife, Jolene. He spends his time away from Microsoft playing soccer, doing triathlons, running, and relaxing with friends and family as much as possible. Chris has a blog at *http://blogs.msdn.com/chrisgarty*.

Chary Gottumukkala joined Microsoft in 2002 as a software architect on the Microsoft Dynamics team, and he works on ERP/CRM frameworks and applications. Chary is passionate about developing software with the often intangible quality attributes, or "-ilities," such as scalability, maintainability, and extensibility. Prior to joining Microsoft, Chary worked on ERP/CRM frameworks and applications at Oracle and PeopleSoft, and on Professional Services Automation (PSA) applications at Niku. Chary has a BSc in electronics from Jawaharlal Nehru Technological University and an MSc in computer science from the Indian Institute of Technology.

Arthur Greef is a principal software architect who has a passion for developing innovative software that simplifies the lives of working people. Arthur has a BSc and an MSc in mechanical engineering from the University of Natal in South Africa, and a Ph.D. in industrial engineering from the University of Stellenbosch in South Africa. He also spent 2 years in an industrial engineering postdoctoral

program at the University of North Carolina in the United States. Arthur has been at Microsoft for 10 years, 3 of which were spent in Denmark working on Microsoft Dynamics AX when it was still called Axapta. Before joining Microsoft, Arthur worked for IBM in New York, developing product configuration technology for PCs. Arthur also spent two years working as chief architect for the RosettaNet Consortium, where he developed XML business collaboration protocols for the information technology industry.

Jakob Steen Hansen is a development manager, currently responsible for the development and architecture of developer tools, business intelligence, application life cycle, upgrade, and customization of Microsoft Dynamics AX. He joined Damgaard Data in 1993 while completing his MSc in computer science and electronic engineering, and contributed to the incubation of the product that later became Microsoft Dynamics AX. Throughout the releases, he has been involved in various aspects of the product, as well as in exploring how technology can bring previously unseen productivity or capabilities to partners and customers who develop solutions by using Microsoft Dynamics AX. For a few years, he worked on an incubation project in the Microsoft Developer Division, which eventually brought him back to the Microsoft Dynamics team.

Jakob worked in Denmark until 2008, when he moved to Seattle with his wife, Lone, and two teenage daughters, Louise and Ida Marie, to explore new facets of working at Microsoft, expanding his personal experience, and realizing new breakthroughs for Microsoft Dynamics and ERP development. He enjoys family life and the outdoors, and because he's an avid engineer, there's always a technical project cooking somewhere.

Kevin Honeyman played a key role in designing the changes to the Microsoft Dynamics AX 2012 user experience. Kevin has worked for Microsoft for 11 years, focusing on simplifying the user experience for various Microsoft Dynamics ERP products. Prior to working at Microsoft, Kevin worked at Great Plains Software, where he designed the developer user experience and user interface controls for the Great Plains Dynamics product. He started his career as a developer at Boeing Computer Services in Seattle, implementing a user interface control library in X Windows and Motif.

As a senior user experience lead at Microsoft, Kevin is passionate about understanding the user's needs and designing experiences that delight the user. He lives in Fargo, North Dakota, with his fiancée, Tiffanie, his son, Jordan, and his future stepchildren, Drue and Isabelle.

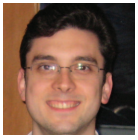
Michael Merz is a program manager for Microsoft Dynamics AX, where he is responsible for the delivery of the Microsoft Dynamics AX services framework and Microsoft Dynamics AX integration capabilities. He has over 15 years of experience in the software industry. Prior to working at Microsoft, Michael held various engineering and management positions in companies including Amazon.com, BEA Systems, and early-stage start-up companies, where he worked on embedded systems, online advertising, social networks, and enterprise software. He has an MSc in computer science from Ulm University, Germany, and lives in Bothell, WA, with his wife, Florina, and his children, Brooke and Joshua.

Amar Nalla is currently a development lead in the Microsoft Dynamics AX product group. He has more than 11 years of experience in the software industry. He started working on the Microsoft Dynamics team during the Axapta 4.0 release. He is part of the foundation team responsible for the Microsoft Dynamics AX server components, and during the past three releases of Microsoft Dynamics AX, he has worked on various components of the server. He maintains a blog at *http://blogs.msdn.com/b/amarnalla/.*

In his spare time, Amar likes to explore the beautiful Puget Sound area.

Parth Pandya is a senior program manager in the Microsoft Dynamics AX product group. For Microsoft Dynamics AX 2012, Parth's area of focus was the new security framework that was built for the release, including the flexible authentication capability and support for Active Directory groups as Microsoft Dynamics AX users. He also contributed to the named user licensing model that was instituted for Microsoft Dynamics AX 2012. Parth has been with Microsoft for over nine years, over five of which were spent working on various releases of the Windows Internet Explorer browser. He particularly enjoyed working as a penetration tester for the number one target of hackers around the world.

Parth swapped the organized chaos of Mumbai, India, for the disorienting tranquility of the Pacific Northwest, where he lives with his wife, Varsha, and three-year-old son, Aarush.



Gustavo Plancarte is a senior software design engineer who joined Microsoft in 2004 after graduating from ITESM in Monterrey, Mexico. He has worked on Microsoft Dynamics AX since version 4.0. On the platform team, he is responsible for driving the common intermediate language (CIL) migration of the X++ programming language, the Software-plus-Services architecture of

the application server, and the batch framework. Gustavo has filed several software-related patents, in areas including garbage collection, incremental generation of assemblies, and batch scheduling and processing. He lives with his wife, Gina, and their sons, Gustavo Jr. and Luis, in Woodinville, WA, where he enjoys spending time working on his yard.

Michael Fruergaard Pontoppidan joined Damgaard Data (which merged with Navision and was eventually acquired by Microsoft) in 1996, after graduating from the Technical University of Denmark. He started as a software design engineer on the MorphX team, delivering the developer experience for the first release of Microsoft Dynamics AX. Today, he is a software architect on the Microsoft Dynamics AX team in Copenhagen. For Microsoft Dynamics AX 2012, Michael primarily focused on the metadata model store, solving problems related to element IDs and the MorphX Development Workspace. In previous releases, he has worked on version control, unit testing, best practices, and the Microsoft Trustworthy Computing initiative, while advocating for code quality improvements through Microsoft Engineering Excellence, tools, processes, and training. Michael frequently appears as a speaker at technical conferences. He lives in Denmark with his wife, Katrine, and their two children, Laura and Malte. His blog is at *http://blogs.msdn.com/mfp*.

Bigyan Rajbhandari is a program manager on the Microsoft Dynamics AX team, working on the security, licensing, and batch framework areas. He has more than seven years of experience in software engineering, consulting, and management, the last four of which have been spent at Microsoft. Prior to his current role, he helped develop a large customer preference management system for Microsoft. He graduated from Drake University in Iowa with a BS in computer science and went on to work for various companies in the Midwest, building custom business applications and customer relationship management (CRM) solutions. Outside of work, he enjoys traveling, hiking, and soccer. He currently lives in Redmond, Washington, with his wife, Jashmin.

Karl Tolgu is a senior program manager for Microsoft Dynamics AX. He is responsible for the development of the business process, workflow, and alert notification framework. Previously, Karl worked on the project accounting modules in Microsoft Dynamics SL and Microsoft Dynamics GP. Since graduating, he has worked in the software industry in both the United Kingdom and the United States and held various software development management positions at Oracle Corporation and Niku Corporation. Karl resides in Seattle with his wife, Karin, and three sons, Karl Christian, Sten Alexander, and Thomas Sebastian.

TJ Vassar has worked in development on various projects at Microsoft for over 13 years, including Microsoft Money, MSN Money, Microsoft Office Accounting, and Microsoft Dynamics AX 2009. Currently, he is a senior program manager on the Microsoft Dynamics AX Business Intelligence team, managing the Reporting framework. Born and raised in Seattle, TJ is married to the woman of his dreams and is a proud father of three. He regularly posts to his MSDN blog (*http://blogs.msnd.com/dynamicsaxbi*) on topics that range from basic development principles to alternate methods of visualizing business insight by using the Reporting framework.

Peter Villadsen is a senior program manager on the Microsoft Dynamics AX X++ language team, developing and maintaining the X++ language stack. After earning his MS in electrical engineering, he started his career by building Ada compilers but quickly became interested in ERP systems, helping to build one for the Apple Macintosh before joining Damgaard Data. There, he helped design and build the first version of what later became the Microsoft Dynamics AX system.

Peter currently lives in Seattle with his wife, Hanne. When not behind the monitor, he enjoys a good game of badminton.

Milinda Vitharana is a senior program manager on the Microsoft Dynamics AX Business Intelligence (BI) team in Redmond, WA. His area of focus is the online analytical framework (OLAP) framework in Microsoft Dynamics AX. Before joining the team in 2008, Milinda spent over 12 years designing, developing, and implementing business intelligence solutions in various industries, including life insurance, financial services, real estate, education, justice, and transportation.

Milinda is passionate about applying BI to solve business problems. He started his career working for an independent software vendor (ISV) developing software solutions in the financial services industry. He then joined a large life insurance company, where he implemented BI solutions. Before joining Microsoft, he worked for a large systems integrator as a BI specialist and consultant. Having seen many applications of BI as an ISV, customer, and partner, he is happy to finally be at the SYS layer.

Milinda is a software engineer and has an MBA in finance. He lives in the greater Seattle area with his wife and two kids.

Christian Wolf is solutions architect and program manager, and is a member of the team that is responsible for the performance and scalability of Microsoft Dynamics AX. Before joining the Microsoft Dynamics AX core development team, he worked as a support, premier field, and escalation engineer, collecting field experience about performance and scalability. He lives in Bellevue, WA, and in his spare time, he enjoys cycling, running, and hiking. Christian's team members maintain a blog about performance and scalability issues at *http://blogs.msdn.com/b/axperf/*.

Contributing authors

Jeff Anderson is a senior software design engineer who joined Microsoft in 2002 after graduating from North Dakota State University in Fargo, North Dakota. He has worked on a variety of Microsoft ERP products, including Microsoft Dynamics GP, Microsoft Dynamics NAV, and Microsoft Dynamics AX. He has been working on Microsoft Dynamics AX since the 2009 release, in the area of global financial management and application performance. He lives in West Fargo, North Dakota, with his wife, Jennifer, and their sons, Nate and Joe.

Wade Baird is a senior software design engineer on the Microsoft Dynamics AX Client Presentation team in Fargo, North Dakota. Wade joined Microsoft in 2001, while completing his final year at North Dakota State University. Since then, he has worked on a variety of Object Relational Mapping (ORM) products, and he began working on Microsoft Dynamics AX for the 2009 release. Since then, he has focused on all of the aspects of the client forms subsystem.

Arijit Basu is a senior solutions architect on the Solutions Architecture team for Microsoft Business Solutions.

Michael Gall is a senior software development engineer on the Microsoft Dynamics AX Costing team at Microsoft Development Center Copenhagen. He joined Microsoft in 2007. Before joining Microsoft, Michael worked with a Microsoft Dynamics AX partner as a solution architect and software development manager, implementing Microsoft Dynamics AX projects in various industries. During the development of Microsoft Dynamics AX 2012, he worked on the lean costing solution and the source document and accounting frameworks. Michael has a Ph.D. in computer science, four master's degrees from the Technical University of Vienna, and an MBA from Copenhagen Business School. Professionally, he is passionate about software architecture and ERP system architecture. He lives in Copenhagen with his children, Laura and Nico. In his spare time, he likes traveling and outdoor activities with his kids.

John Healy is a principal software architect in the Microsoft Dynamics AX product group that focuses on global financial management. He is responsible for the overall vision and adoption of the architecture for global financials and works with a range of Microsoft Dynamics AX architects and technical leaders to ensure consistent direction and adoption across the Microsoft Dynamics AX applications. He has over 32 years of experience in accounting, supply chain, and manufacturing application development. He has worked in a variety of technical and leadership roles. He joined Microsoft in 2001 through the Great Plains Software acquisition. He is a graduate of the University of Minnesota, Twin Cities. He lives in Lake Elmo, Minnesota, with his wife, Jackie.

John is an editor and regular contributor to the Microsoft Dynamics AX Global Financial Management team blog at *http://blogs.msdn.com/b/ax_gfm_framework_team_blog/.*

Vanya Kashperuk is a senior software development engineer in Test on the Supply Chain Management team at Microsoft Development Center Copenhagen. He has been working on Microsoft Dynamics AX since 2004 and has been at Microsoft in Denmark for the last four years. Vanya writes a blog about Microsoft Dynamics AX, which you can read at *http://www.kashperuk.blogspot.com*.

Vanya has a master's degree in computer science from the National Technical University of Ukraine, which is also where he met his wife, Valeriia.

Outside of work, Vanya enjoys all kinds of team sports, sightseeing in countries around the world, photography (as part of his sightseeing trips), and computer games, especially the new Kinect Sports!

Vanya and his wife live in a peaceful area of Charlottenlund, just north of Copenhagen.

Ievgenii Korovin is a software design engineer on the Supply Chain Management team at Microsoft Development Center Copenhagen. He has been working on Microsoft Dynamics AX since 2006, and he is mainly responsible for the architecture and functionality within the Inventory management, Warehouse management, and Product information management areas. Ievgenii has a master's degree in computer science from the National Technical University of Ukraine. He lives in Copenhagen with his wife, Tamara, and their young daughter, Alisa. In their free time, he and his family enjoy outdoor activities, especially skiing, biking, and hiking.

Ievgenii writes a blog about Microsoft Dynamics AX, which you can read at *http://blogs.msdn.com/dynamicsaxscm*.

Maciej Plaza is a software development engineer in Test on the Microsoft Dynamics AX Inventory team. He received an MSc from Poznan University of Technology, and during his time at the university, he actively sought out interesting algorithmic problems, engaging in research projects in cooperation with partners from both industry (Volkswagen) and academia (University

of Nottingham). After graduating in 2007, he started working as Software Development Engineer for Microsoft SQL Server, tackling the challenges of storing unstructured data, working on FILESTREAM, Remote BLOB Storage, and FileTable features. After almost three years, he decided to move back to Europe and joined the Microsoft Dynamics AX team. For the Microsoft Dynamics AX 2012 release, his primary focus was ensuring the quality of the Product information management functionality. Driven by his passion for quality, he also got involved in improving the test tools and the applied processes, to ensure that even higher quality can be achieved in the future.

Maciej lives in Copenhagen with his wife, Anna. In his spare time, besides spending time with his family, he enjoys pursuing his interests in photography and music.

Anders Tind Sørensen joined Microsoft in 2006 as a software development engineer for the Manufacturing team. He has focused primarily on discrete production, master planning, and the resource scheduling engine, but has also been deeply involved in the integration of the Process manufacturing industry module. He has 10 years of ERP development and implementation experience, and is proud to be a geek. Anders lives in Denmark with his girlfriend, Nena.

Manoj Swaminathan is a principal development manager based in Redmond, WA. He joined Microsoft 4 years ago, after working at Oracle, and has spent more than 12 years working on ERP application development for financials. He was responsible for leading global financial management development for the financial foundation, such as the source document and accounting frameworks, multiple ledgers, and globalization/localization of Microsoft Dynamics AX 2012. Currently, he is leading efforts to drive application life cycle management for Microsoft Dynamics AX, RapidStart Services, and setup/deployment initiatives for online and on-premise solutions.

Robin Van Steenburgh joined the Microsoft Dynamics AX team in 2005; she currently enjoys creating developer samples and documentation with the software development kit (SDK) team based in Redmond. After graduating from the University of Toronto, Robin worked as a software developer for several oil companies and a software startup called Sierra Geophysics. Robin joined Microsoft in 1997 and has worked on teams delivering MSN, Site Server, Commerce Server, and Microsoft Learning products. Her favorite role prior to joining Microsoft Dynamics AX was as an acquisitions editor for Microsoft Press. She is a Microsoft Certified Technology Specialist for Microsoft Dynamics AX 2009 and Microsoft Dynamics AX 2012. Robin was responsible for the developer documentation for services and the Application Integration Framework (AIF) on MSDN for Microsoft Dynamics AX 4.0 and Microsoft Dynamics AX 2012. She also maintains the MSDN Developer Center for Microsoft Dynamics AX, and occasionally blogs at *http://blogs.msdn.com/b/aif/*. In her free time, she takes ballet classes and supports the Seattle Sounders.

What do you think of this book?

We want to hear from you!

To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Tell us how well this book meets your needs—what works effectively, and what we can do better. Your feedback will help us continually improve our books and learning resources for you.

Thank you in advance for your input!

Microsoft® Press