

THE EXPERT'S VOICE® IN SQL SERVER

THIRD EDITION

Pro SQL Server 2012 Reporting Services

*CREATE BUSINESS VALUE
FROM POWERFUL AND EFFECTIVE
REPORTING ON MICROSOFT'S
ENTERPRISE-LEVEL PLATFORM*

Brian McDonald, Shawn McGehee, and Rodney Landrum

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Contents at a Glance

■ About the Authors	xv
■ About the Technical Reviewers	xvi
■ Acknowledgments	xvii
■ Introduction	xix
■ Chapter 1: Introducing the Reporting Services Architecture.....	1
■ Chapter 2: Report Authoring: Designing Efficient Queries	19
■ Chapter 3: Introduction to Reporting Services Design with SQL Server Data Tools..	39
■ Chapter 4: Laying Out a Report.....	61
■ Chapter 5: Implementing Dashboard-Style Report Objects	89
■ Chapter 6: Building Reports.....	125
■ Chapter 7: Using Custom .NET Code with Reports	185
■ Chapter 8: Deploying Reports	213
■ Chapter 9: Rendering Reports from .NET Applications	243
■ Chapter 10: Managing Reports	279
■ Chapter 11: Securing Reports.....	329
■ Chapter 12: Delivering Business Intelligence with SSRS.....	361
■ Chapter 13: Creating Reports Using Report Builder 1.0, 2.0, and 3.0	401
■ Index	483

Introduction

At its core, the process of designing reports hasn't changed substantially in the past 20 years. The report designer lays out report objects, which contain data from a known source of data, in a design application such as Reporting Services, Business Objects Reports, or Microsoft Access. He or she then tests report execution, verifies the accuracy of the results, and distributes the report to the target audience.

Sure, there are enough differences between design applications to mean that the designer must become familiar with each particular environment. However, there's enough crossover functionality to make this learning curve small. For example, the SUM function is the same in Business Objects Reports as it is in Microsoft Access as it is in Structured Query Language (SQL).

With Microsoft SQL Server 2012 Reporting Services (referred to as SSRS throughout the book), there is, again, only a marginal difference in the way reports are designed from one graphical report design application to another. So, if you do have previous reporting experience, your learning curve for SSRS should be relatively shallow. This is especially true if you come from a .NET environment, because the report designer application for SSRS 2012 is Visual Studio 2010 or the application included with SQL Server 2012, SQL Server Data Tools (SSDT), formerly known as Business Intelligence Development Studio (BIDS). We use BIDS and SSDT interchangeably throughout the book, with most references using BIDS. We have done this mainly because of the role that Reporting Services plays in the Business Intelligence stack of products with SQL Server, but also for readers who may be using prior versions of Reporting Services like SSRS 2008 R2.

Having said all this, several differences set SSRS apart from other reporting solutions:

- It provides a standard reporting platform based on Report Definition Language (RDL), which is the XML schema that dictates the common structure of all SSRS reports. This allows for report creation from any third-party application that supports the RDL schema.
- SSRS is an integral part of the SQL Server 2012 release.
- SSRS offers features out of the box that in other products would be expensive additions to a basic deployment. These features include subscription services, report caching, report history, and scheduling of report execution.
- SSRS can be extended with third party add-ons, custom code, and compiled DLL's.
- SSRS, being a Web-based solution, can be deployed across a variety of platforms.
- SSRS also allows for easy integration with Microsoft's Collaboration Software for the Enterprise: SharePoint 2010.

This book was written in parallel with a real SSRS deployment for a health-care application, so it covers almost every design and deployment consideration for SSRS, always from the standpoint of how to get the job done effectively. You'll find step-by-step guides, practical tips, and best practices, along with code samples that you'll be able to modify and use in your own SSRS applications.

Who This Book Is For

We coauthored the book with the intention of demonstrating how to use SSRS from multiple vantage points. As reporting architects and report developers, we go through the report design and deployment processes using standard SSRS tools such as Report Designer in BIDS and Report Manager. We also show how developers can extend SSRS by creating custom Windows and Web Forms applications.

Prerequisites

The core software that has been used in the examples throughout this book are:

- Microsoft SQL Server 2012
- Microsoft Visual Studio 2010 – used in chapters 7, 8, 9, and 10
- Microsoft SharePoint 2010 – used in chapter 12 with SSRS integration
- Microsoft SQL Server 2008 R2 – used in chapter 13 for ad hoc reporting using Report Models

Each of the aforementioned software is required if you, the reader, have the desire to follow along with the examples throughout the book. Most of the examples were built using SQL Server 2012, but with the exception of chapters 7, 8 and 9, they can be performed on SQL Server 2008 R2.

Downloading the Code

In this book, we use a subset of real databases designed for a health-care application that some of us developed over the years. You can find all of the supporting materials (databases, the data mart database, and cube file used in Chapter 12, the completed RDL files, queries, stored procedures, and .NET application projects, as well as full installation instructions) in the Source Code/Download section of the Apress Web site (www.apress.com). With so many other books with similar titles having existed over the years, it may be easier to find this book by using its ISBN number. The 13-digit industry standard ISBN number for this book is 978-1-4302-3810-2.

Contacting the Authors

Should you have any questions regarding any section in the book, please feel free to contact us via our email or twitter accounts. We would love to hear that you have purchased our book, so please feel free to tweet us. We sincerely hope that you get the enjoyment out of reading the book that we had in writing it for you.

Brian K. McDonald
bmcdonald@sqlbigEEK.com
[@briankmcdonald](https://twitter.com/briankmcdonald)

Shawn McGehee
shawnnwf@gmail.com
[@SQLShawn](https://twitter.com/SQLShawn)

Rodney Landrum
rodneylandrum@hotmail.com
@SQLBeat

Introducing the Reporting Services Architecture

Microsoft's 2003 announcement that it was going to release SQL Server Reporting Services (SSRS) as a SQL Server 2000 add-on stirred up a frenzy of excitement. The product was originally slated for release with SQL Server 2005, so the early release was a welcome event for many. Our software development company decided to embrace SSRS early on and was fortunate to work with Microsoft during the beta phases. In January 2004, the month Microsoft's released SSRS to manufacturing (RTM), we deployed it immediately. We intended to migrate all of our existing reports (which we had developed on as many as five reporting applications and platforms over the previous ten years) to SSRS. We can sum up the reason for the seemingly rapid decision in one word: standardization.

Just as Microsoft wanted to create an industry standard with Report Definition Language (RDL), the Extensible Markup Language (XML) schema that dictates the common structure of all SSRS reports, we wanted to provide a standard reporting solution to our customers. Even in the first version of the product, SSRS delivered almost all the features we needed. Thanks to its extensibility via SSRS's Web service, we could programmatically add other features that weren't already directly supported. In addition, Microsoft was committed to enhancing SSRS for years to come. Some of the features released in the 2005 edition were client-side printing, interactive sorting capabilities, and an ability to define multivalued parameters. There was also a move forward in the self-service business intelligence (BI) arena with Microsoft's first ad hoc Report Builder ClickOnce application.

Microsoft's next release was SSRS 2008. The new release brought on many long-awaited enhancements to include modifications of its architecture, completely revamped report designer, and 2008 R2 brought us significant design updates to the built-in Report Manager application. With the vast updates implemented in the 2008 release, SSRS has taken its place as a key SQL Server component in Microsoft's business intelligence suite of products alongside SQL Server Integration Services (SSIS) and SQL Server Analysis Services (SSAS). Nobody could now think of Reporting Services as just an add-on.

The new features in SSRS 2008 and SSRS 2008 R2 pushed the technology one step further into becoming the reporting development environment of choice for programmers and designers, especially those already skilled with Visual Studio (VS) and Visual Basic .NET (VB.NET). As they were for its predecessors, SSRS 2005, SSRS 2008, and SSRS 2008 R2, the long-awaited features for SSRS 2012 are mostly driven by direct feedback from the user community. Throughout the book, we will demonstrate each of the new features released in 2008, 2008 R2, and 2012 as we show how to design professional reports, applications, and solutions built on Microsoft's BI initiatives. We will focus on SSRS as a whole, building on features from each version from 2000 to 2012; however, we will point out which features are new to SSRS 2008 R2 and SSRS 2012.

Understanding the Benefits of SSRS

Our company based its decision to migrate immediately to SSRS on the following perceived benefits for the company and for our customers:

Standard platform: As well as providing a standard realized with the RDL, our development teams had been using VS .NET as their main development environment. Because SSRS reports were currently developed within this platform, we wouldn't need to purchase additional development software. Our clients would need to purchase only a low-cost edition of a designer—VB.NET, for example—to gain the benefit of developing their own custom reports. In SQL Server 2005, Microsoft included the Business Intelligence Development Studio (BIDS) as a free, alternative report designer. This free development environment has been available with SQL Server ever since, but Microsoft has recently renamed it as SQL Server Data Tools (SSDT). Throughout this book, we will use BIDS and SSDT interchangeably. The BIDS environment runs in the shell of Visual Studio (devenv.exe) and, at the time of writing, is based on Visual Studio—VS 2008 for 2008 and 2008 R2, and VS 2010 for the latest release, SQL Server 2012. Anybody who learns to design reports with BIDS will have the advantage of a consistent interface when they move to the full version of Visual Studio, and will need no additional training.

Cost: SSRS is an integral part of SQL Server 2012 and is available in many editions, from Express Advanced to Enterprise in 2008 and even Datacenter edition in 2008 R2. However, because SQL Server 2012 has done away with Datacenter edition, the most feature-rich edition will once again be Enterprise. When you purchase SQL Server, you get SSRS as well. See a complete list of SQL Server 2012 features at <http://tinyurl.com/SQL2012Features>.

Web-enabled: Because SSRS is a Web-based reporting solution; a single deployed report is accessible to a variety of clients, from the browser to custom Windows Forms. In addition, because reports are primarily accessed via Hypertext Transfer Protocol (HTTP) or HTTP Secure (HTTPS), you can view reports from any location that has access to the SSRS Web server. Unless you have a thick client application that requires local reports to be deployed with the application, you can have one central repository for reports to be consumed across the organization.

Customizable: SSRS provides a .NET Web service as a front end, programmatically accessible to extend the delivery of reports beyond the browser. As .NET programmers, we knew we would want to build custom applications to render reports where we could control the look and feel of the report viewer. We show one such application in Chapter 7, which covers report rendering.

Subscriptions: SSRS subscription abilities gave a huge advantage for our company and our clients, as report delivery by e-mail or file-sharing, as well as off-peak processing, were now possible. We show how to set up two different kinds of subscriptions, standard and data-driven, in Chapter 8.

As you'll see, SSRS is a full reporting solution that encompasses many levels of professional expertise, from report design to database administration. In many organizations, especially small- to medium-sized ones, information technology (IT) professionals are asked to perform many jobs. They

write a query and design a report in the morning, perform database backups or restores in the afternoon, and update all the systems before heading home in the late hours of the evening. Sometimes even until the early hours of the next day! But we are sure that we're not the only ones that take such pride in our jobs and always striving to exceed the needs of the business.

Throughout each of our careers, we have all worn many hats in the companies which we have poured our time and devotion to over the years. We have been entrenched in every deployment phase from internal deployments to externally facing web application deployments to our clients, from simple implementations to advanced ones which extended Reporting Services capabilities. By developing efficient stored procedures, thoroughly testable security mechanisms, as well as building and maintaining well designed reports, we have witnessed the day-to-day operation of SSRS from many perspectives.

We have also been responsible for our company's overall strategy for building solutions to analyze and transform data gathered through both our own and third-party applications. To that end, an essential part of our jobs over the years has been to integrate SSRS into the overall BI strategy that incorporated the following:

- Disparate data sources such as Analysis Services Cubes and SQL Server relational databases
- Applications and tools such as Microsoft Excel and Business Scorecards
- Document management systems such as Microsoft SharePoint Portal Server

We'll dive into the details of such integration projects in Chapter 12, which is devoted to BI. We will also explore one of the key advancements of SSRS 2008 R2 and 2012, which is a tighter integration with SharePoint portal server, to the point that SSRS content can now be directly deployed, managed, and viewed all within SharePoint. We'll also show you how sections of reports can be created and served as web parts.

SSRS represents another world, not often seen by an administrator using standard management tools. This world is the domain of the software developer who can extend and control SSRS programmatically, building custom report viewers and deployment applications. In this book, as you work through each step of building a reporting solution for healthcare professionals, you'll see how an administrator can accomplish the task with built-in tools, as well as how a developer can create an application to provide enhanced functionality.

SQL Server 2008 R2 and 2012 Reporting Services Enhancements

There have been many major additions to Reporting Services since its initial release in 2005, but let's look at some of the most significant enhancements made to the SSRS technology in SQL Server.

Report Builder/Data Modeler

The Report Builder application, a feature introduced in SSRS 2005, is a local ad hoc report-designing application intended for use more by report consumers than by report developers. An administrator familiar with the source data creates the business logic and underlying data structures as a data model. With the Report Builder application, the user can create and publish reports based on available models. Microsoft designed Report Builder 2.0, released in SSRS 2008, for Microsoft Ribbon technology, much like Microsoft Word and Microsoft Excel, and it was a significant improvement on Report Builder 1.0. Each enhancement provided a richer development environment and additional content sources, such as Oracle and Analysis Services Cubes. As if that wasn't enough, Report Builder 3.0 made its first appearance with SSRS 2008 R2, with its new data visualization report items and cached result sets.

Chapter 13 tells you how to build and deploy a data model, as well as how to create reports with the Report Builder 1.0, Report Builder 2.0, and Report Builder 3.0 applications.

SSRS 2012 Integration with Microsoft Office SharePoint

While SharePoint integration was available with the use of SharePoint controls in previous versions of SSRS, SSRS 2012 takes the integration several steps further. By using SSRS 2012 in SharePoint Integration Mode, users can deploy, manage, and deliver reports and report objects, like web parts, data sources, and models, even entire dashboards or portals, all within the SharePoint environment. In addition, the deployed reports inherit the native features of SharePoint, such as workflow capabilities and the ability to check in and check out reports, and report change notification. We will demonstrate this tighter integration with SharePoint in Chapter 12.

Tablix Report Properties

As the name suggests, the Tablix properties first seen in SSRS 2008 combine two existing report controls, Table and Matrix. This combination gives developers a more flexible tool when creating reports. The availability of multiple columns and rows blends the static nature of the Table control with the dynamic nature of the Matrix. Reports can now accommodate multiple parallel rows and column members at each level, independent of each other but using the same aggregate calculations. In previous editions of this book, we provided workarounds to combining tables and matrices by embedding one within the other. In Chapter 4, we will explore the true power of the new Tablix control properties for the List, Table, and Matrix controls.

Enhanced Charting and Report Item Visualizations

From the beginning, SSRS offered charts and visualizations natively in reports. These charts, while versatile, were somewhat limited in scope. Much, if not all, of the functionality in the charting aspects of previous versions of SSRS could be easily duplicated in Microsoft Excel. In fact, the charting was almost identical. SSRS 2008 provided several charting and graphical data-visualization enhancements vital for the sound BI reporting solution of which SSRS is a pivotal component. New charting elements such as range, polar, radar, funnel, and pyramid are available, as well as many new “gauges” delivered with the acquisition of Dundas reporting controls for SSRS.

SSRS 2008 R2 included several eagerly-awaited report item visualizations to enable the creation of a more sophisticated dashboard look and feel. One is the Map control that can display data from a geospatial data result set or an Environmental Systems Research Institute, Inc. (ESRI) shape file. Other great additions include Data Bars, Sparklines, and Indicators. We will explore several of these new visualizations as we incorporate them into reports in Chapter 5.

Enhanced Performance and Memory Management

Microsoft reengineered the report engine in SSRS 2008 to lessen the memory footprint for reports at the server level, speeding delivery of reports to end-user applications. This enhancement also resolves the contention that arose when long-running, large reports and smaller, non-memory-bound reports processed simultaneously.

Embeddable SSRS Controls

The ability to embed controls in custom applications makes it easier for developers to integrate SSRS into their projects. Since the release of SQL Server 2005, the Visual Studio environment has included distributable controls that you can use for Windows Forms development and ASP.NET Web Forms development. These controls provide additional benefits to developers, such as the ability to render reports while disconnected from the SSRS. We will cover updated SSRS controls in Chapter 9.

HTML Text Formatting

Aside from the change from dual to single service architecture and the ability to export to Microsoft Word format, text formatting is probably one of the most significant advancements of SSRS 2008. In previous versions of SSRS, in-line formatting of textual content, for example for a form letter, was not possible. For example, if you wanted to have a single textbox contain some text in regular font, but wanted to bold or italicize other sections of the text, you wouldn't be able to do it. Textbox report items in SSRS 2008, SSRS 2008 R2, and SSRS 2012 allow for normal and rich-text modes and allow formatting in the same way as a word processor does. You can create a placeholder to allow a limited subset of some HTML and style tags. The text formatting can combine both literal text and data source text for mail merge and template reports. We will demonstrate the full use of this feature by creating a custom form letter style report in Chapter 6.

Microsoft Word Rendering

Since the first version of SSRS, you could export any report to Microsoft Excel. While this was an important capability, not being able to export to other Microsoft Office formats, such as Word, was a limitation. Developers often want to create reports using the rich text formats found in today's modern word processors. By combining SSRS's ability to design custom reports from multiple data sources with Word's ability to provide rich formatting, SSRS 2008 overcomes significant limitations of its predecessors. Another limitation was that report users could not export into 2007 formats. Excel 2003 has a limitation of 65,536 rows and 256 columns, but one of SSRS 2012's new rendering enhancements enables us to export to Word and Excel 2007-2010 formats, so we can now store 1,048,576 records and 16,384 columns on one sheet of an Excel workbook.

Report Parts

If you're like us, you have probably wanted to create little reusable objects that could be incorporated in more than one report. Until SSRS 2008 R2, you could do this only by creating reports that could be embedded into other reports as subreports. Now, you can publish individual sections of reports, like a Tablix containing the top 10 employees by sales totals or a Sparkline showing the customer complaints trend for the current year. Any report item like these can be deployed to a ReportServer or SharePoint server and reused by end users using ad hoc tools like Report Builder or SharePoint. Furthermore, report developers can use these report parts to reduce duplicate efforts for reports that need the data represented in the same fashion. A very useful feature is that if the report part is modified by a user with appropriate permissions, the consumers of that report part are notified about the update and they can choose to refresh their report or to leave it as it was.

Lookup Functions

Before 2008 R2, the need often arose to find a value in a different data region in the same report. Since the data in both of the tables could be linked by a common field, one workaround was simply to gather the data in the source query by joining the tables together. Often, this may perform better on the backend, but sometimes you need to look up values in another data region, so Microsoft has added three lookup functions to Reporting Services: Lookup, LookupSet and MultiLookup.

Shared Datasets

Before moving on from the upgrades released with SSRS 2008 R2 and 2012, let's briefly talk about shared datasets. This feature, added in 2008 R2, allows you to create a dataset that can be consumed among other reports. Imagine you have created a project with 50 reports, about 10 of them with a parameter for all of the countries throughout the world. Thinking of manageability, you designed this dataset to be loaded by making a call to a stored procedure. In previous releases of SSRS, you would have needed to create a dataset for every report that needed this parameter, so any change that affected them all would have had to be made in them all, one at a time. From 2008 R2 onward, we can create a shared dataset and use it across reports. A change in that single dataset updates all the reports that need that change.

SSRS and Business Intelligence

SSRS is just one component of Microsoft's BI platform. We'll now cover other new features and enhancements since SQL Server 2008 that will form an integral part of your overall reporting solution.

Business Intelligence Development Studio and SQL Server Data Tools

Business Intelligence Development Studio (BIDS) is a limited version of Visual Studio 2008, included with the SQL Server 2008 base installation. In SQL Server 2012, the report designer takes on a new name, SQL Server Data Tools (SSDT), and we now have the Visual Studio 2010 shell rather than VS 2008. With SSDT and BIDS, developers can create entire projects for each of the supported components of SQL Server 2012, including SSIS, SSAS, and of course SSRS. We will use SSDT throughout the book (except in Chapter 13, where we use Report Builder to show you how to design and deploy SSRS reports and Analysis Services projects). Note that SSDT and BIDS both use the devenv executable and as such, can be used interchangeably.

SQL Server Management Studio (SSMS)

With the release of SQL Server 2008, Microsoft continued to build on its management platform with SQL Server Management Studio (SSMS). Microsoft has taken a big step toward consolidating, in a single environment, many of the tools that in previous versions of SQL Server would have been executed individually. SSMS replaces Enterprise Manager and Query Analyzer, offering a much more elaborate set of tools for creating and managing SQL Server objects and queries. In addition to managing SQL Server and Analysis Services servers, administrators can use SSMS to manage instances of their SSRS reporting servers. We have heard in the SQL Server community that Management Studio will run in the Visual Studio shell, but for now, anyway, it still runs with `ssms.exe`. However, SSMS users will now be able to undock windows and have them on multiple monitors as Visual Studio developers have done for some years.

Throughout the book, we will show you how to use both SSMS and Report Manager for various tasks. For example, we will show you how to use SSMS to test query performance and the browser-based Report Manager to view published reports, set security permissions, and create subscriptions. Although the two applications share functionality for managing SSRS, Report Manager is often preferable to SSMS because it can perform many more administrative tasks and does not require a local installation. You can access Report Manager from a browser anywhere on your network, but you would need access to the installed SQL Server 2012 client tools to use SSMS.

SSRS Architecture

You've probably heard the expression that the devil is in the details. You'll be drilling into those details throughout the book, right down to the data packets that SSRS constructs, as you explore each aspect of SSRS from design to security. For now, let's pull back to a broader vantage point—the 10,000-foot view if you will—and look at the three main components that work together to make SSRS a true multi-tier application: the client, the report server, and the SQL Server report databases. Figure 1-1 shows the conceptual breakdown of the three component pieces.

The data source and the SSRS databases, ReportServer and ReportServerTempDB, are separate entities. The data source is the origin of the data that will populate the reports, while the report server databases store metadata and execution information about the reports. Both the data source and the report server databases can physically be located on the same SQL Server, assuming the data source is a SQL Server database. The data source can be any supported data provider, such as SQL Server, Oracle, Lightweight Directory Access Protocol (LDAP), Microsoft SharePoint List, SQL Azure and Analysis Services. It's possible to configure a single server to act as both the SSRS report server web service and report server database as well as the data source server. However, this isn't recommended unless you have a small user base. We'll show you how to monitor the performance of the SSRS configuration and build a small Web farm, post-installation, in Chapter 10.

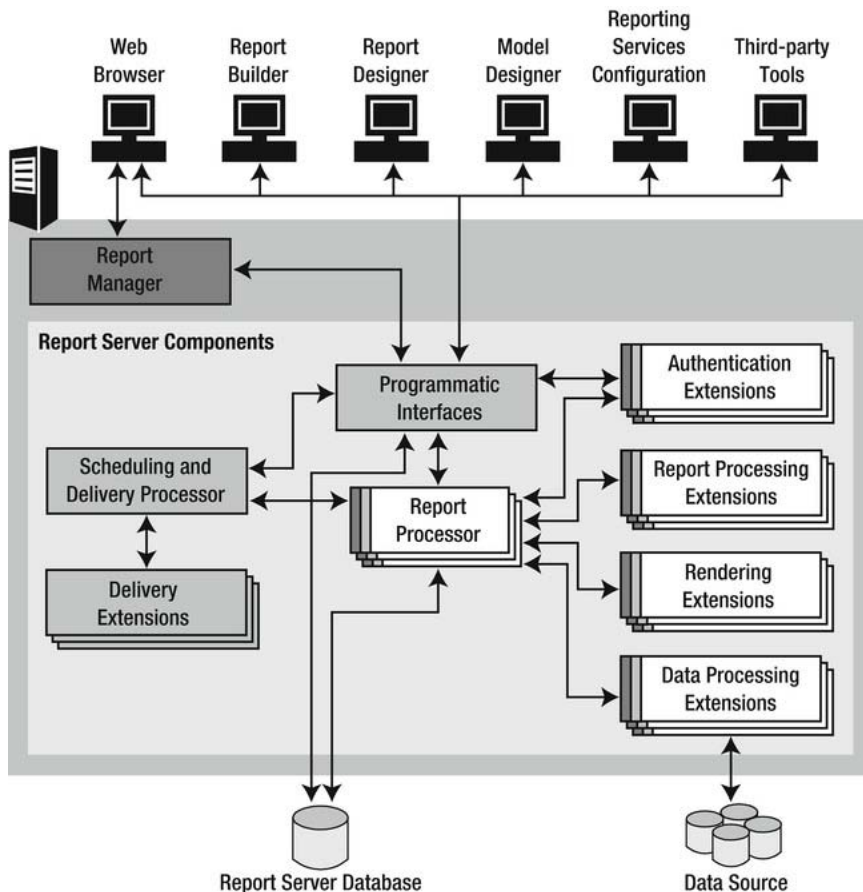


Figure 1-1. SSRS components

SSRS Databases

SSRS is added as an option during the SQL Server installation process. The SSRS native installation creates two databases that are used to store report metadata and manage performance:

ReportServer: This is the primary database that stores all the information about reports that was originally provided from the RDL files used to create and publish the reports to the ReportServer database. In addition to report properties (such as data sources) and report parameters, ReportServer also stores folder hierarchy and report execution log information.

ReportServerTempDB: This database houses cached copies of reports that you can use to increase performance for many simultaneous users. By caching reports using a nonvolatile storage mechanism, you make sure they remain available to users even if the report server is restarted.

Database administrators can use standard tools to back up and restore these two databases. An additional database might be added after the initial installation of SSRS: the `RSEExecutionLog` database. This stores more detailed information about report execution, such as the user who ran the report, the time of execution, and performance statistics. We'll cover creating the `Pro_SSRSExecutionLog` database and discuss report execution logging in detail in Chapter 10.

■ **Note** When configuring Reporting Services to run in SharePoint integrated mode for 2012, an extra database is installed for Alerting. Note also that the default database names are slightly different as they will have a unique identifier appended to `ReportingService_`, assigned when creating the instance of Reporting Services on the SharePoint site. For example, in SharePoint integrated mode, `ReportServerTempDB` becomes something like `ReportingService_14214aae2b5d4f0d888289011932bmcdTempDB`. We'll look at SharePoint integrated mode in chapter 12.

The SSRS Report Server

The SSRS report server plays the most important role in the SSRS model. Working in the middle, it's responsible for every client request to render a report or to perform a management request, such as creating a subscription. You can break down the report server into several subcomponents by function:

- Programming interface
- Authentication Layer (new in SSRS 2008)
- Report processing
- Data processing
- Report rendering
- Report scheduling and delivery

SSRS Web Service Interface

The programming interface, exposed as .NET Web service application programming interfaces (APIs) and uniform resource locator (URL) access methods, handles all incoming report and management requests from clients. Depending on the type of request, the programming interface either processes it directly by accessing the `ReportServer` database or passes it off to another component for further processing. If the request is for an on-demand report or a snapshot, the Web service passes it to the Report Processor before delivering the completed request to the client or storing it in the `ReportServer` database.

■ **Note** On-demand reports are rendered and delivered directly to the client, while snapshots are reports that are processed at a point in time and delivered to the client through e-mail or via file shares, or (if configured) directly to a printer.

Authentication Layer

SSRS 2005 relied heavily on the authentication methodology of Internet Information Services (IIS), since SSRS and IIS were interdependent. With the exception of SSRS in SharePoint integrated mode, no SSRS versions since 2008 are tied to IIS. SSRS now uses `Http.sys` directly, as well as SQL Server's native network components, so SSRS's architecture has been redesigned to include its own authentication layer, which we will cover in Chapter 11.

The Report Processor

The Report Processor component is responsible for all report requests. Like the programming interface, it communicates directly with the ReportServer database to receive the report definition information that it then uses to combine with the data returned from the data source, which is accessed via one of the data processing extensions.

Data Processing

Reporting Services supports twelve data processing extensions to connect to data sources. These are:

- SQL Server
- Oracle
- OLE DB
- OLEDB-MD
- ODBC
- XML
- SAP BI NetWeaver
- Hyperion Essbase
- Teradata
- Microsoft SQL Azure (SQL in the Cloud)
- Microsoft SQL Server Parallel Data Warehouse
- Microsoft SharePoint List

When the data processing component receives the request from the Report Processor, it initiates a connection to the data source and passes it the source query. Data is returned and sent back to the

Report Processor, which then combines the elements of the report with the data returned from the Data Processor extension.

Report Rendering

The combined report and data is handed off to the rendering extension component to be stored in an intermediate format called Report Page Layout (RPL). The RPL is then turned into one of several supported or custom formats, based on the rendering type specified by the client (we cover rendering in depth in Chapter 8):

- *HTML*: Default rendering format, supporting HTML versions 4.0 and 3.2.
- *Portable Document Format (PDF)*: Format used to produce print-ready reports using Adobe Acrobat Reader. SSRS doesn't require that you have an Adobe license to render in PDF, which is a great benefit to customers. All you need is a PDF reader.
- *Excel 2002 and 2003*: Service Pack 1 of SSRS supports Excel 97 and later. As discussed previously, SQL Server 2012 supports exporting in 2007-2010 (.xlsx) compressed format to allow more rows and columns.
- *XML*: Other applications or services can use reports that are exported to XML.
- *Comma-separated values (CSVs)*: By rendering to a CSV file, you can further process the report by importing it into other CSV-supported applications such as Microsoft Excel.
- *MIME HTML (MHTML)*: You can use this format, also known as a *Web archive*, to deliver reports directly in e-mail or to deliver them for storage, because the report contents, including images, are embedded within a single file.
- *Tagged Image File Format (TIFF)*: Rendering image files using TIFF guarantees a standard view of the report, as it's processed the same way for all users, whatever their browser settings or versions.
- *Microsoft Word*: Standard Microsoft Word document export is now included in SSRS 2008. Both 97-2003 (.doc) and 2007-2010 (.docx) compressed formats are available in SSRS 2012.
- *ATOM*: This format can be consumed by ATOM-compliant client applications such as PowerPivot and SharePoint.
- *NULL*: The NULL rendering extension isn't actually a format like the others, but can be used to cache results of reports. The next time the report is requested; it is pulled from cache and rendered significantly quicker. This is especially useful if you have a larger report that takes an abnormally long time to render. You will see this extension as a delivery format when creating subscriptions.

Scheduling and Delivery

If the request from the client requires a schedule or delivery extension, such as a snapshot or subscription, the programming interface calls the Scheduling and Delivery Processor to handle the request. You can generate and deliver report snapshots, based on a user-defined or shared schedule, to

one of two supported delivery extensions: an e-mail or a file share. Note that SSRS uses the SQL Server Agent to create the scheduled job. If the SQL Server Agent isn't running, the job won't execute. We'll cover creating subscriptions and snapshots based on shared schedules in Chapter 10.

Client Applications

SSRS includes several client applications that use the SSRS programming interface, including Web service APIs and URL access methods, to provide front-end tools for users to access both SSRS reports and configuration tools. These tools provide report server management, security implementation, and report-rendering functionality. The tools are as follows:

- *Report Manager*: This browser-based application ships with SSRS and provides a graphical interface for users who need to view or print reports, or to manage report objects for their workgroups or departments. We describe Report Manager in detail in Chapter 10, which covers managing SSRS.
- *BIDS or SSDT*: This tool provides an integrated environment for developing SSRS reports. We introduce BIDS or SSDT in Chapters 3-5 and step through building reports in this environment in Chapter 6 and throughout the book.
- *Command-line utilities*: You can use several command-line tools to configure and manage the SSRS environment, including `rs`, `rsconfig`, `rskeymgmt` and `rsactivate`.
- *Report Builder 3.0*: This enhanced application was primarily developed to give business users the ability to create ad hoc reports. Nearly all of the features available in BIDS are also available in Report Builder 3.0.
- *Custom clients*: These .NET Windows Forms and Web applications call the SSRS Web service to perform such tasks as rendering reports and managing report objects. SSRS includes sample application projects that you can compile and run to extend the functionality provided by the main tools listed earlier. In Chapters 8 and 9, we show you how to develop your own custom applications: a report viewer and a report publisher.
- *Reporting Services Configuration Manager*: SSRS for SQL Server 2008 included an enhanced Reporting Services Configuration Manager designed specifically to change many of these properties in a graphical environment, including setting up the SSRS environment for offline or disconnected reporting.

When thinking of a Web-based application, the natural inclination is to think *Web browser*. Even though other front-end tools, such as SSMS and Visual Studio, connect to the report server, a Web browser plays an important role in providing the graphical interface for users, who can use Report Manager to view or print reports or remotely manage the report server for their workgroups or departments.

Report Manager

Within Report Manager, users can render reports, create report subscriptions, modify the properties of report objects, and configure security, as well as perform a host of other tasks. Users can access Report Manager by simply opening their Web browser and navigating to a URL of the form `http://Servername/Reports`. Figure 1-2 shows Report Manager in action, with a listing of reports in a folder deployed specifically for clinicians.

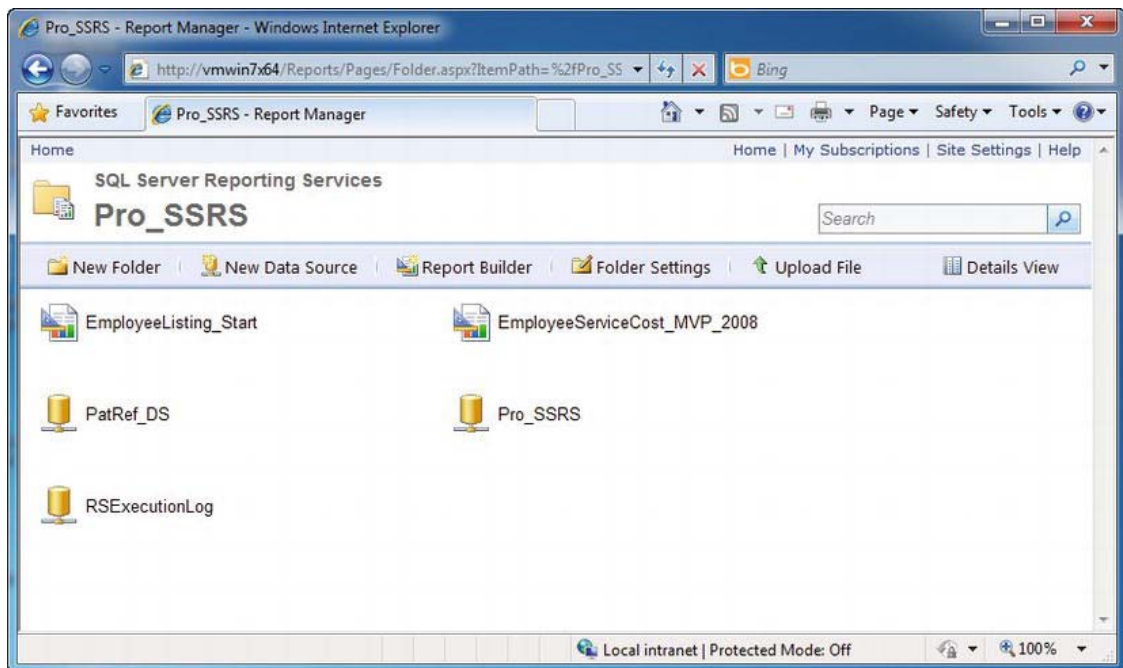


Figure 1-2. The Web-based Report Manager application

Business Intelligence Development Studio (BIDS) and SQL Server Data Tools (SSDT)

The browser is only one of several clients that can use the SSRS Web service. In fact, BIDS is a client when it interacts with the Web service to deploy reports and data sources. BIDS offers a graphical design environment for report developers to produce the RDL files that SSRS uses for deploying and rendering reports.

■ **Note** Because RDL is a defined standard, you can use any design application that supports the creation of RDL files. Other third-party report designers are available, and many more are said to be in development.

By defining the base URL and folder name in a BIDS report project, you can deploy created RDL files directly to the report server while in design mode. The base URL is of the form `http://Servername/ReportServer`. If you have SSRS configured to run under a port other than the default, 80, specify `http://ServerName:PortNumber/ReportServer`. We'll cover the entire BIDS design environment in Chapters 3-5, including most available report objects. We'll also describe the RDL schema that defines every aspect of an SSRS report. Figure 1-3 shows the BIDS design environment, also called an integrated development environment (IDE), with a report loaded in design mode.

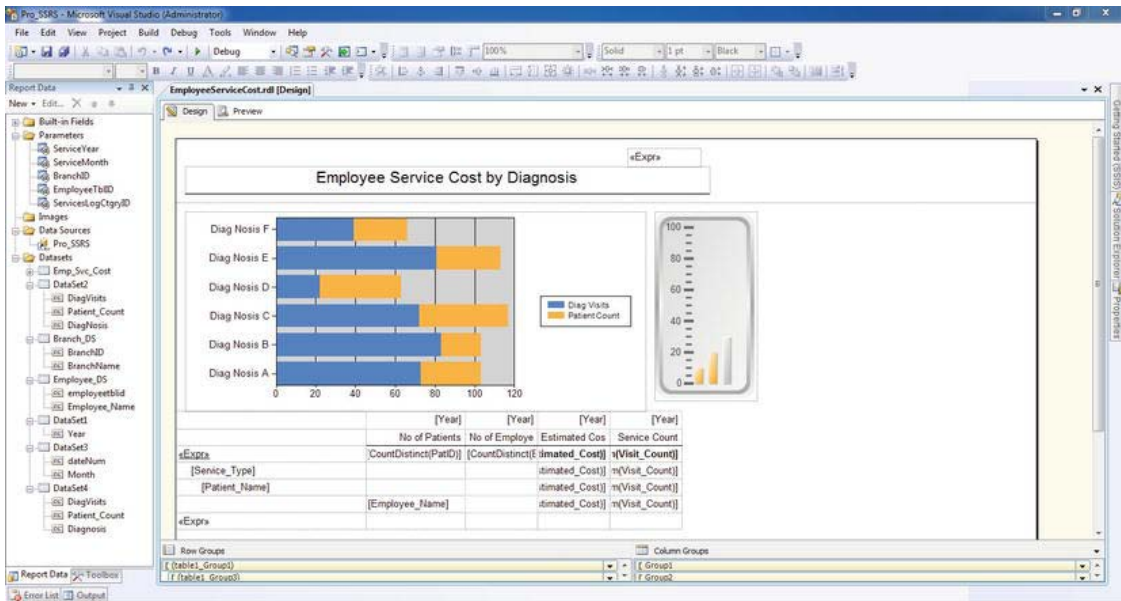


Figure 1-3. BIDS environment

Command-Line Utilities

In addition to graphical applications such as BIDS and SSMS, SSRS provides several command-line utilities that are considered Web service clients. The tools have the added benefit of being automated by using built-in task scheduling in Windows. SSRS includes four main command-line utilities:

- **rs:** Processes reporting services script (RSS) files that are written with VB.NET code. Because you can access the full SSRS API via the code in the script, all SSRS Web service methods are available.
- **rsconfig:** Configures the authentication methods and credentials for connecting SSRS to the ReportServer database. The rsconfig utility also sets the unattended SSRS execution credentials.
- **rskeymgmt:** Manages the encryption keys that SSRS uses to store sensitive data securely, such as authentication credentials. Chapter 11 covers the use of rskeymgmt.
- **rsactivate:** Adds another instance of Reporting Services to a Web Farm, and is useful for replacing a corrupted instance.

Report Builder

The Report Builder application develops reports in an environment much like other Microsoft Office products, with a ribbon style interface. Report Builder 3.0, released with SSRS 2008 R2, provides nearly all of the capabilities of the Report Designer within BIDS. You can install it using standard ClickOnce

technologies from within the browser or as a thick client application to make it available under the Windows Start menu. Figure 1-4 shows an example of a report from Chapter 6, loaded in the Report Builder.

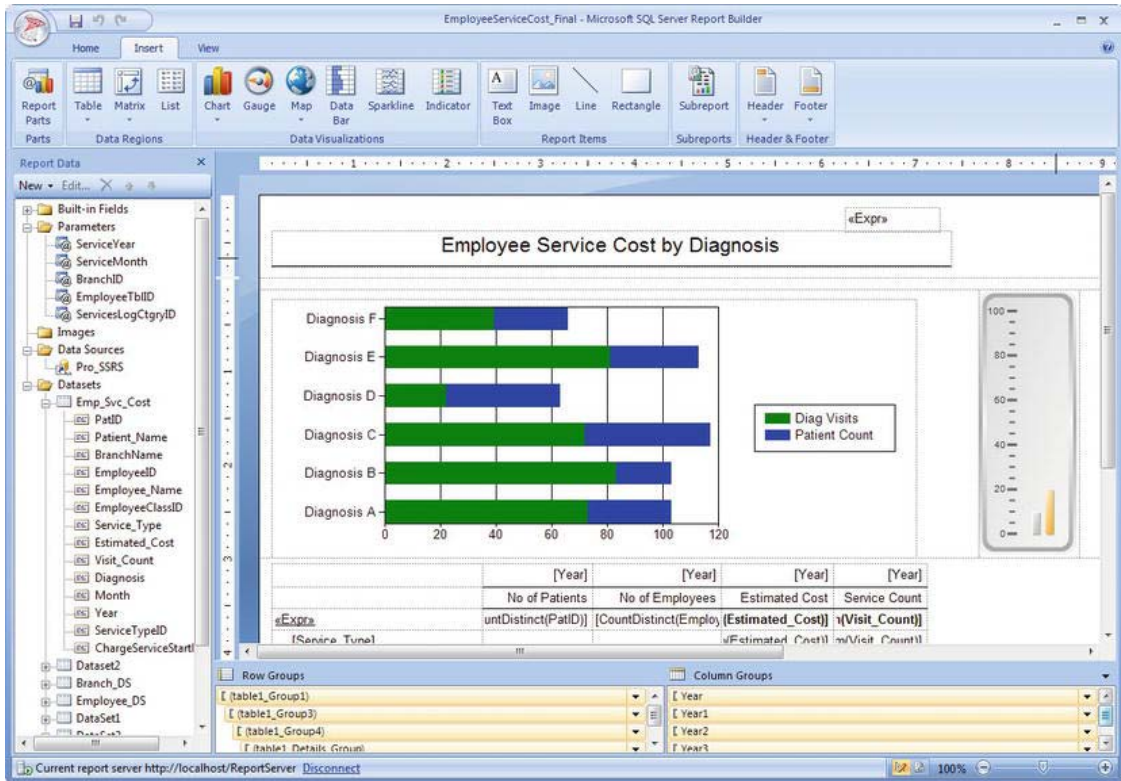


Figure 1-4. Report Builder 3.0

Custom Clients

The final types of clients are those custom designed to access the SSRS Web services. Between all of the authors of this book, we have built several such applications for many organizations, which use both the report viewer and the report publisher. Third-party commercial applications provide extended functionality, for instance RS Scripter, which assists in scripting and managing various report server catalog items; it can extract RDLs from a server, and manage subscriptions, roles, and even migration from one server to another.

Installing and Configuring

You can install Reporting Services—like Analysis Services, Integration Services, and Notification Services—as part of the main SQL Server installation. Before you install Reporting Services 2012, you must know whether you are going to run in native or SharePoint integrated mode. With earlier releases of Reporting Services, you could change mode after installation by using the Reporting Services

Configuration Manager, but with the 2012 release, you can only configure Reporting Services in SharePoint integrated mode at SQL Server installation, as the application will be installed and configured as a native SharePoint service.

The components installed depend on the mode your organization chooses to implement. For instance, when installing in native mode, components include the Reporting Services Web service, ReportServer databases, and Report Manager web application. Once you have installed Reporting Services, you can disable some features if you do not need them. You might choose to do this, for example, if you wanted to use a custom application to interface with the ReportServer rather than the built in Report Manager web application. Installed client components include the administrative command-line tools mentioned previously, such as `rs` and `rsconfig`, as well as documentation, samples, and the Reporting Services Configuration Manager mentioned earlier.

If you use the default configuration settings for native mode during the installation of Reporting Services, you will use the RS Configuration Manager to set up or change various other settings related to the instance. Some tasks to perform are:

- Choosing security settings, such as whether the report server will use HTTP, HTTPS, or both.
- Configuring a Simple Mail Transfer Protocol (SMTP) mail server to handle the delivery of subscriptions.
- Backup and restore the encryption key used when you move a report server to a new machine.
- Set up Web Service and Report Manager URLs and define the ports they run on. This gives you the ability to run multiple reporting services instances on one machine.

As mentioned, once you have installed SSRS, you can modify some of the configuration settings. For example, after reviewing performance data, you might decide that the report server needs to connect to an existing Web farm. You can perform this task using the `rsconfig` utility or by using the graphical Reporting Services Configuration Manager. You can reconfigure the security settings or the mail server by directly modifying the `RSReportServer.config` file. We'll cover using these tools, modifying the configuration file settings, and gathering performance measures in Chapters 10 and 11.

If you choose to use SharePoint integrated mode, the configuration settings are managed completely within SharePoint's Central Administration. We will cover more of SharePoint integration in Chapter 12.

Deploying SSRS Securely

Security ranks as one of the highest priorities for businesses today. Providing customers and employees with a secure and reliable computing environment is not only good practice, but also in many cases a requirement, mandated by stringent federal regulations. In our case, this means adherence to the Health Insurance Portability and Accountability Act (HIPAA). This requires policies and procedures to be in place that guarantee confidential patient information is not only securely transmitted, but also accessible only by those with the authority to view it. To that end, we have to ensure that the data we transmit over a network connection, especially the Internet, is encrypted at its source.

SSRS is a role-based application that provides access to the objects it stores through the use of defined roles, such as content browsers who may only view reports and report data. The roles are associated with Windows-based login accounts, so SSRS relies on Windows as its primary source of authentication. It is possible to extend the security model for SSRS to support other methods of authentication, such as forms-based authentication, whereby users can log in with accounts maintained

outside Windows to access the report server. Since SSRS has multiple authentication points—namely, at the report server level through `http.sys` and the data-access level, SQL, or Windows authentication—specific security risks exist when altering the default Windows roles-based security model. For one, `http.sys` would need to be set up to allow explicit access and custom validation of a user's credentials. Another risk is that SSRS can support only one security extension at a time. In other words, a single SSRS report server can be extended to support a non-default authentication model or remain as a default Windows authentication, but cannot take advantage of both models simultaneously. Depending on your level of need for custom security—say, for example, you need to deploy SSRS on an Internet-facing server, or your application already supports forms authentication, and it would be too difficult to work within the constraints of Windows authentication—then you might need to consider a custom security extension. Our needs were such that we could easily incorporate SSRS into an existing Windows authentication model.

Another method of dealing with security is through a Windows or Web-based application that has its own authentication layer. Using the `ReportViewer` control within the Visual Studio designer for Windows and Web forms, you can use an application as a portal into the report server. As long as the application handles security, you can give the application server access to the needed objects within Reporting Services by using an Active Directory computer account like `DOMAIN\ServerName$`.

In this book, we'll cover two deployment scenarios:

- Intranet deployment using Virtual Private Network (VPN) and firewall technologies to allow access to the SSRS report server
- An Internet-hosted application that uses Terminal Services to connect securely to an SSRS report server

In Chapter 11, we'll walk you through securing the SSRS deployment models with technologies that provide the required encryption levels and user authentication. In addition to the two models that we cover, we briefly discuss ways to integrate a forms-based authentication method allowing clients to connect directly to SSRS via the Internet.

Summary

Having created and deployed numerous projects with SSRS for SQL Server 2005, 2008 and 2008 R2, we have been anxiously awaiting, along with the rest of the SQL Server community, the release of SQL Server 2012. As you work through the book, we will point out the enhancements released with SSRS 2008 R2 and SSRS 2012. However, our main aim, as with the other editions of the book, is to show you how to take advantage of advanced features, providing useful examples, enabling you to put SSRS to work in a real-world environment where the user of the reports and applications that you deploy will have the final say on the solution's success.

Report Authoring: Designing Efficient Queries

SSRS provides a platform for developing and managing reports in an environment that includes multiple data sources of information. These data sources can include both relational data (for example, SQL Server, Oracle, MySQL, and so on) and non-relational data (for example, Active Directory, LDAP stores, and Exchange Server). Standards such as ODBC, OLE DB, and .NET facilitate the retrieval of data from these disparate data stores, so SSRS can access the data as long as your system has the relevant drivers. In the SSRS report design environment, configuring a dataset that drives the report content is the first step of the design process.

However, before we introduce the many elements of the report design environment, it's important to begin with the heart of any data-driven report—whether it's Business Objects Reports, SSRS, or Microsoft Access—and that is the query. With any report design application, developing a query that returns the desired data efficiently is the key to a successful report.

In this chapter, we will describe the following:

The healthcare database that is the target of the reporting queries in this book: You must understand the design of the data before you can design efficient queries. We'll also describe an easy way to familiarize yourself with your data when the full schema details are not available.

How to design basic but effective SQL queries for reporting purposes: We'll create queries based on real-world applications, the kind that report writers and database administrators create every day.

How to use SSMS to gauge query performance: The initial query defines the performance and value of the report, so it's important to understand the tools required to create and test the query to ensure that it's both accurate and tuned for high performance.

How to transform the optimized query into a parameterized, stored procedure: This gives you the benefit of pre-compilation and cached execution plans for faster performance and the benefit of the procedure being centrally updated and secured on SQL Server.

Introducing the Sample Relational Database

Throughout the book, we'll show you how to design and deploy a reporting solution and build custom .NET SSRS applications for an SQL Server-based healthcare application, using relational tables and stored procedures. The application was originally designed for home health and hospice facilities that offer clinical care to their patients, typically in their homes. Our example for this book, the Online

Transactional Processing (OLTP) database, powers this application, capturing billing and clinical information for home health and hospice patients. The database that we will use is called Pro_SSRS and is available for download in the Source Code/Download area of the Apress Web site (<http://www.apress.com>), together with instructions in the ReadMe.txt file on how to restore the database in preparation for use in this and subsequent chapters.

Introducing the Schema Design

Over the years, developers have added features to the application and altered the database schema many times to accommodate the new functionality and to capture required data. This data is needed not only to perform operational processes such as creating bills and posting payments to the patient's account, but also to provide valuable reports that show how well the company is serving its patients. Because these types of healthcare facilities offer long-term care, our customers need to know if their patients' conditions are improving over time and the overall cost of the care delivered to them.

The database designed for the application consists of more than 200 tables and has many stored procedures. In this book, you'll use a subset of that database to learn how to develop reports that show the cost of care for patients. You'll use eight main tables for the queries and stored procedures throughout the book, and you'll begin using some of them to build reports as you work through the next three chapters. These tables are as follows:

Trx: The main transactional data table that stores detailed patient services information. We use the term *services* to refer to items with an associated cost that are provided for patient care.

Services: Stores the names and categories for the detailed line items found in the Trx table. Services could be clinical visits such as a skilled nurse visit, but could also include billable supplies, such as a gauze bandage or syringes.

ServiceLogCtgry: The main grouping of services that are similar and provide a higher-level grouping. For example, all visits can be associated with a "Visits" ServiceLogCtgry for reporting.

Employee: Stores records specific to the *employee*, in this case a clinician or other staff member such as a chaplain visiting a hospice patient. An employee is assigned to each visit stored in the Trx table.

Patient: Includes demographic information about the patient receiving the care. This table, like the Employee table, links directly to the Trx table for detailed transactional data.

Branch: Stores the branch name and location of the patient receiving the care. *Branches*, in the sample reports, are cost centers from which visits and services were delivered.

ChargeInfo: Contains additional information related to the individual Trx records that is specific to charges. Charges have an associated charge, unlike payments and adjustments, which are also stored in the Trx table.

Diag: Stores the primary diagnoses of the patient being cared for and links to a record in the Trx table.

Figure 2-1 shows a graphical layout of the eight tables and how they're joined.

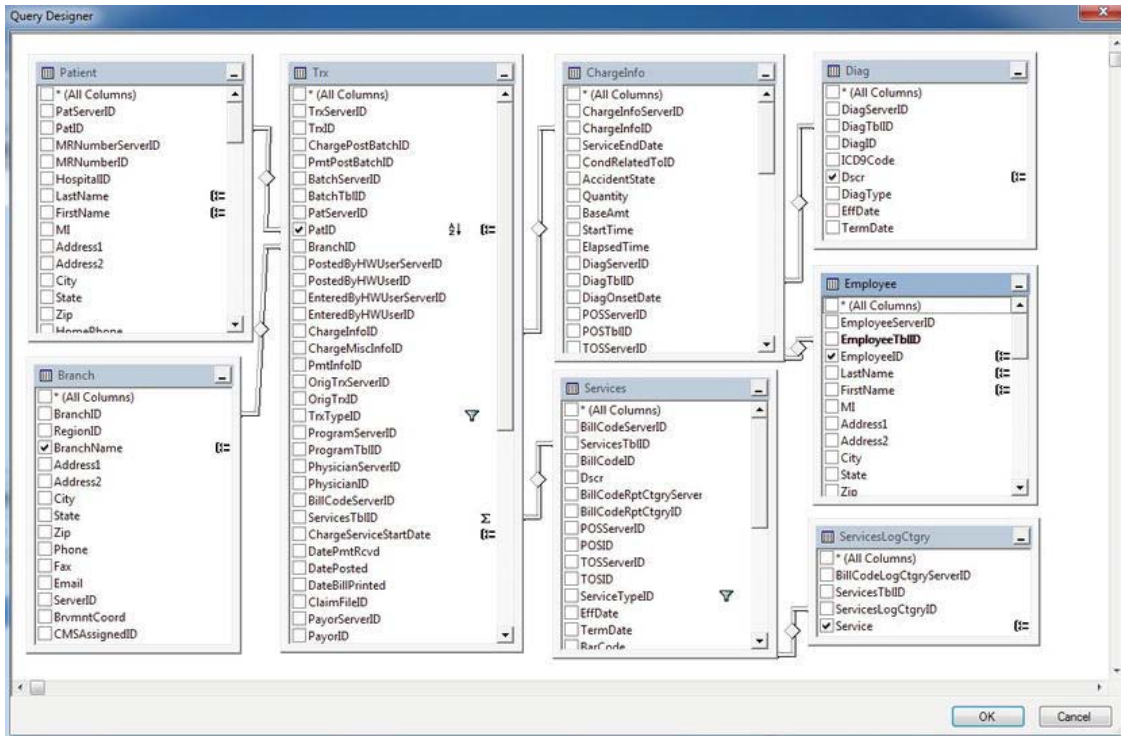


Figure 2-1. Viewing the sample application's database tables

Knowing Your Data: A Quick Trick with a Small Procedure

For every report writer, familiarity with the location of the data in a given database can come only with time. Of course, having a database diagram or schema provided by a vendor is a useful tool, and we have the luxury of that here, but this isn't always available. One day, faced with the dilemma of trying to find the right table for a specific piece of missing data, we decided to put together a stored procedure, which we named `sp_FieldInfo`. It returns a list of all the tables in a specific database that share field names, typically the primary or foreign key fields. For example, in the healthcare database, if you want a list of tables that contain the `PatID` field (the patient's ID number that's used to join several tables), you would use the following command:

```
sp_FieldInfo PatID
```

The output would be similar to that shown in Table 2-1.

Table 2-1. Output of sp_FieldInfo

Table Name	Field Name
PatCertDates	PatID
PatDiag	PatID
PatEMRDoc	PatID
Trx	PatID
Patient	PatID
Admissions	PatID

Armed with this information, you could at least deduce that, for example, the patient’s diagnosis information is stored in the PatDiag table. However, table and field names aren’t always intuitively named. When we encounter a database such as this from time to time, we can run a SQL Server Profiler trace and perform some routine tasks on the associated application, such as opening a form and searching for an identifiable record to get a starting point with the captured data. The Profiler returns the resulting query with table and field names that we can then use to discern the database structure.

■ **Tip** SQL Server Profiler is an excellent tool for capturing not only the actual queries and stored procedures executing against the server, but also the performance data, such as the duration of the execution time, the central processing unit (CPU) cycles and input/output (I/O) measurements, and the application that initiated the query. Because you can save this data directly to an SQL table, you can analyze it readily, and it even makes a good source of data for a report in SSRS.

Listing 2-1 displays the code to create the sp_fieldinfo stored procedure. You can find the code for this query in the code download file CreateFieldInfo.sql in the SQL Queries folder.

Listing 2-1. Creating the sp_FieldInfo Stored Procedure

```
IF EXISTS (SELECT * FROM INFORMATION_SCHEMA.ROUTINES
           WHERE ROUTINE_NAME = 'sp_FieldInfo'
           AND SPECIFIC_SCHEMA = 'dbo')
    DROP PROCEDURE [dbo].[sp_FieldInfo]
GO
CREATE PROCEDURE [dbo].[sp_FieldInfo]
(
    @column_name VARCHAR(128) = NULL
)
```

```

AS
SELECT
    TABLE_NAME AS [Table Name]
    , RTRIM(COLUMN_NAME) AS [Field Name]
FROM
    INFORMATION_SCHEMA.COLUMNS
WHERE
    COLUMN_NAME LIKE '%' + @column_name + '%'

```

Introducing Query Design Basics

Whether you're a seasoned pro at writing SQL queries manually through a text editor or someone who prefers to design queries graphically, the result is what matters. Accuracy, versatility, and efficiency of the underlying query are the three goals that designers strive to achieve. Accuracy is critical, but a query that performs well and is versatile enough to be used in more than one report makes the subsequent report design task much easier. For scalability and low response times, efficiency is paramount. A great report that takes 15 minutes to render will be a report your users rarely run. Keep the following goals in mind as you begin to develop your report queries:

The query must return accurate data: As the query logic becomes more complex, the chance of inaccuracy increases with extensive criteria and multiple joins.

The query must be scalable: As the query is developed and tested, be aware that its performance might change radically as the load increases with more users. We cover performance monitoring with simulated loads in Chapter 10. However, in this chapter we'll show how to use tools to test query response times for a single execution in order to improve performance.

The query should be versatile: Often a single query or stored procedure can drive many reports at once, saving on the time it takes to maintain, administer, and develop reports. However, delivering too much data to a report at once, to support both details and a summary, can affect performance. It's important to balance versatility with efficiency.

Creating a Simple Query Graphically

Query design typically begins with a request. As the report writer or database administrator (DBA), you're probably often tasked with producing data that's not available through the standard reports often delivered with third-party applications.

Let's begin with a hypothetical scenario. Say you receive an e-mail that details a report to be created and deployed for an upcoming meeting. It has already been determined that the data is unavailable from any known reports, yet you can derive the data using a simple custom query.

In this first example, you'll look at the following request for a healthcare organization:

Deliver a report that shows the ten most common diagnoses by service count.

Assuming you are familiar with the database, begin the query design process in SSMS, either graphically or by coding the query with the generic query designer. Both methods are available within SSMS.

■ **Note** We'll cover setting up the data source connection required for building an SSRS report in Chapter 3. For now, you'll connect directly to the data with the available query design tools within SSMS. It is important to mention that though you are designing the query within SSMS, similar tools are available within the BIDS environment so that you can create your queries at the same time you create your report. We chose SSMS in this case because it contains lists of database objects that you may need to reference as you begin to develop the query.

We'll show you how to design the query with the graphical tool to demonstrate how the underlying SQL code is created. You can access the graphical query designer by right-clicking anywhere in the new query window within SSMS and selecting Design Query in Editor (see Figure 2-2).

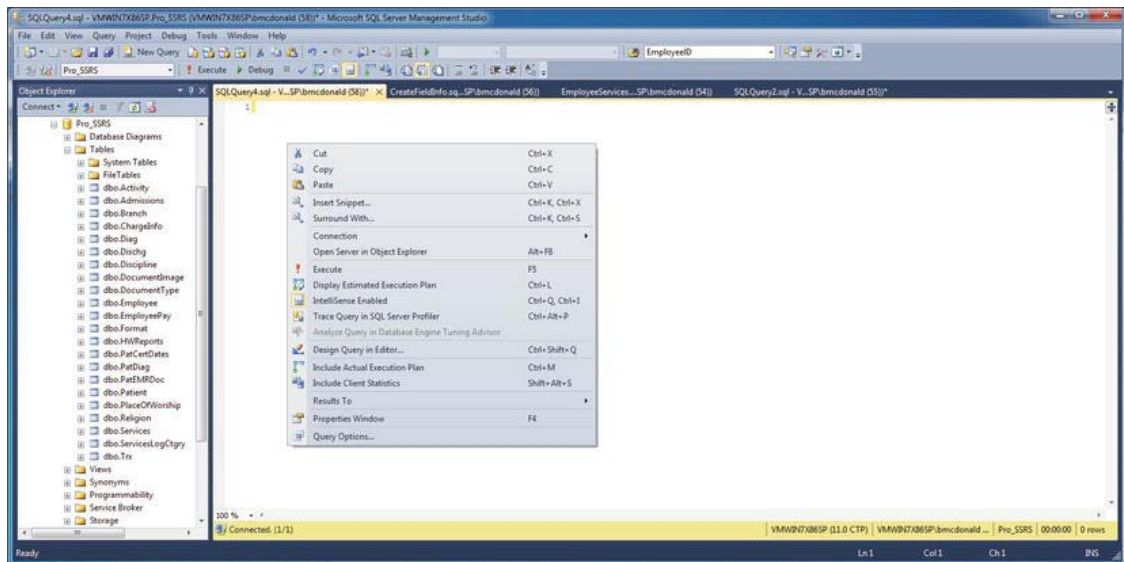


Figure 2-2. Accessing the query design tool in SSMS

After you open the query designer, you can perform tasks such as adding and joining additional tables and sorting, grouping, and selecting criteria using the task panes (see Figure 2-3).

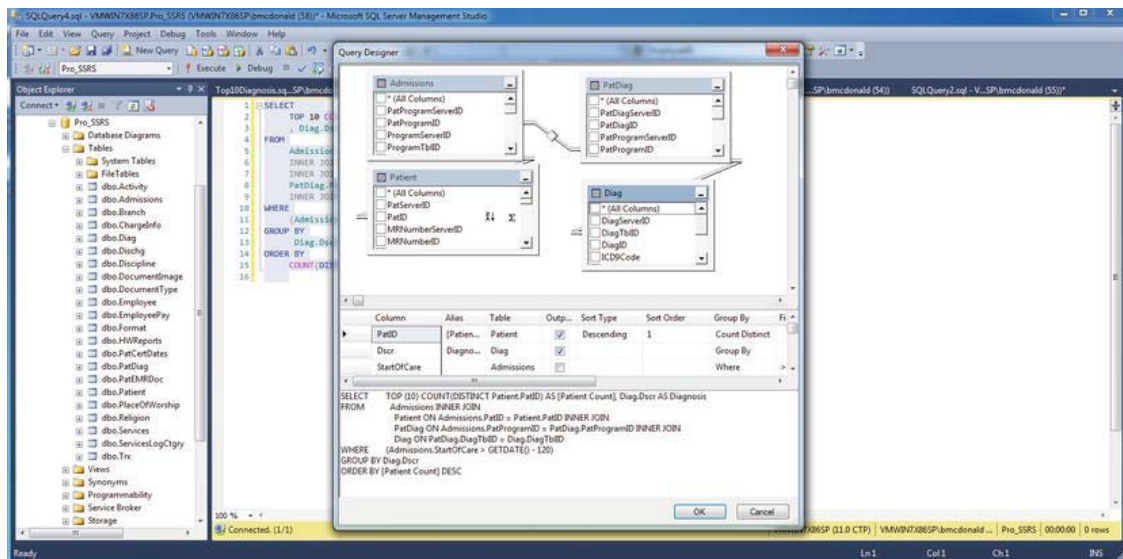


Figure 2-3. Working with the graphical query designer in SSMS

This initial query is a relatively simple one; it uses four tables joined on relational columns. Through the graphical query designer, you can add basic criteria and sorting, and you can select only two fields for the report: a count of the patients and a specific medical diagnosis. You can make the Sort Type of the count Descending so that you can see the trend for the most common diagnoses. You can directly transport the SQL query to a report, and we'll show you how to do that in Chapter 6. Listing 2-2 shows the query produced. You can find the code for this query in the code download file `Top10Diagnosis.sql` in the Source Code/Download area of the Apress Web site (<http://www.apress.com>) in the SQL Queries folder.

Listing 2-2. The SQL Query Produced Using the Graphical Query Designer to Return the Top Ten Patient Diagnoses

```
SELECT TOP 10
    COUNT(DISTINCT Patient.PatID) AS [Patient Count]
    , Diag.Dscr AS Diagnosis
FROM
    Admissions
    INNER JOIN Patient ON Admissions.PatID = Patient.PatID
    INNER JOIN PatDiag ON Admissions.PatProgramID = PatDiag.PatProgramID
    INNER JOIN Diag ON PatDiag.DiagTblID = Diag.DiagTblID
GROUP BY
    Diag.Dscr
ORDER BY
    COUNT(DISTINCT Patient.PatID) DESC
```

Table 2-2 shows the output of this query.

Table 2-2. Sample Output from the Top Ten Diagnoses Query

Patient Count	Diagnosis
152	ABNORMALITY OF GAIT
107	BENIGN HYPERTENSION
83	BENIGN HYP HRT DIS W CHF
77	PHYSICAL THERAPY NEC
64	DECUBITUS ULCER
59	DMI UNSPF UNCNTRLD
55	CHR AIRWAY OBSTRUCT NEC
52	ABNRML COAGULTION PRFILE
49	DMII UNSPF NT ST UNCNTRL
43	CONGESTIVE HEART FAILURE

This particular aggregated query has a small result set. Even though it’s potentially working with tens of thousands of records to produce the resulting ten records, it runs in less than a second. This tells you that the query is efficient, at least in a single-user execution scenario.

This type of query is designed to deliver data for quick review by professionals who will make business decisions from the results of the summarized data. In this example, a healthcare administrator will notice a demand for physical therapy and might review the staffing level for physical therapists in the company. Because physical therapists are in high demand, the administrator might need to investigate the cost of caring for physical therapy patients.

Creating an Advanced Query

Next, we’ll show how to design a query that reports the cost of care for the physical therapy patients. The goal is to make the query and subsequent report flexible enough to include other types of medical services that can be analyzed as well, not only physical therapy. This query requires more data for analysis than the previous one did. Because you’ll process thousands of records, you need to assess the performance impact.

The design process is the same. Begin by adding the necessary tables to the graphical query designer and selecting the fields you want to include in the report. The required data output for the report needs to include the following information:

- Patient name and ID number
- Employee name, specialty, and branch

- Total service count for patient by specialty
- Diagnosis of the patient
- Estimated cost
- Dates of services

Listing 2-3 shows the query to produce this desired output from the healthcare application. You can find the code for this query in the code download file `EmployeeServices.sql` in the SQL Queries folder.

Listing 2-3. Employee Cost Query for Healthcare Database

```

SELECT
    Trx.PatID
    , RTRIM(RTRIM(Patient.LastName) + ', ' + RTRIM(Patient.FirstName)) AS [Patient Name]
    , Employee.EmployeeID
    , RTRIM(RTRIM(Employee.LastName) + ', ' + RTRIM(Employee.FirstName)) AS [Employee
Name]
    , ServicesLogCtgy.Service AS [Service Type]
    , SUM(ChargeInfo.Cost) AS [Estimated Cost]
    , COUNT(Trx.ServicesTblID) AS Visit_Count
    , Diag.Dscr AS Diagnosis
    , DATENAME(mm, Trx.ChargeServiceStartDate) AS [Month]
    , DATEPART(yy, Trx.ChargeServiceStartDate) AS [Year]
    , Branch.BranchName AS Branch
FROM
    Trx
    JOIN ChargeInfo ON Trx.ChargeInfoID = ChargeInfo.ChargeInfoID
    JOIN Patient ON Trx.PatID = Patient.PatID
    JOIN Services ON Trx.ServicesTblID = Services.ServicesTblID
    JOIN ServicesLogCtgy ON Services.ServicesLogCtgyID =
ServicesLogCtgy.ServicesLogCtgyID
    JOIN Employee ON ChargeInfo.EmployeeTblID = Employee.EmployeeTblID
    JOIN Diag ON ChargeInfo.DiagTblID = Diag.DiagTblID
    JOIN Branch ON TRX.BranchID = Branch.BranchID
WHERE
    (Trx.TrxTypeID = 1) AND (Services.ServiceTypeID = 'v')
GROUP BY
    ServicesLogCtgy.Service
    , Diag.Dscr
    , Trx.PatID
    , RTRIM(RTRIM(Patient.LastName) + ', ' + RTRIM(Patient.FirstName))
    , RTRIM(RTRIM(Employee.LastName) + ', ' + RTRIM(Employee.FirstName))
    , Employee.EmployeeID
    , DATENAME(mm, Trx.ChargeServiceStartDate)
    , DATEPART(yy, Trx.ChargeServiceStartDate)
    , Branch.BranchName
ORDER BY
    Trx.PatID

```

The alias names identified with AS in the SELECT clause of the query should serve as pointers to the data that answers the requirements of the report request. Again, knowing the schema of the database

that you'll be working with to produce queries is important, but for the sake of the example, the joined tables are typical of a normalized database where detailed transactional data is stored in a separate table from the descriptive information and therefore must be joined. The Trx table in Listing 2-3 is where the transactional patient service information is stored, while the descriptive information of the specialty services such as "Physical Therapy" is stored in the Services table.

Other tables, such as the Patient and Employee tables, are also joined to retrieve their respective data elements. You use the SQL functions COUNT and SUM to provide aggregated calculations on cost and service information, and RTRIM to remove any trailing spaces in the concatenated patient and employee names. You can use the ORDER BY PATID clause for testing the query to ensure that it's returning multiple rows per patient as expected. It isn't necessary to add the burden of sorting to the query. As you'll see in the next chapters, sorting is handled within the report. Dividing the load between the SQL Server machine that houses the report data and the report server itself is important and often requires performance monitoring to assess where such tasks as sorting, grouping, and calculating sums or averages for aggregated data will be performed. If the report server is substantial enough to shoulder the burden and is less taxed by user access than the actual data server is, it might be more efficient to allow it to handle more of the grouping and sorting loads. More often than not though, it is considered best practice to have the relational database engine perform as much of the work as possible to alleviate some of the load on the report server.

Testing Performance with SQL Server Management Studio (SSMS)

Now that you have developed the query, you'll look at the output to make sure it's returning accurate data within acceptable time frames before moving on to the next phase of development. Figure 2-4 shows the results of the output from SSMS and the time it took to execute the query. You can directly modify the query further in SSMS if you want to. However, one of the best features of SSMS you'll notice is the ability to view quickly both the number of records returned and the execution time. Once you've done that, the next step is to create the stored procedure.

You now have the data the way you want, and the query is executing in an average of one second. To verify the execution times, run the query 15 times in sequence from two different sessions of SSMS. Execution times will vary from zero to two seconds for each execution. For 3,924 records, which is the number of records the query returns, the execution time is acceptable for a single-user execution. However, you need to improve it before you create the stored procedure, which you will want to scale out to accommodate hundreds of users, and begin building reports.

Looking at the Execution Plan tab in SSMS will give you a better understanding of what's happening when you execute the query. In SSMS, click the Display Estimated Execution Plan button on the toolbar. When the query is executed, the Execution Plan tab appears in the Results pane. Alternatively, if you just want to see the execution plan without returning results, just press CTRL+L.

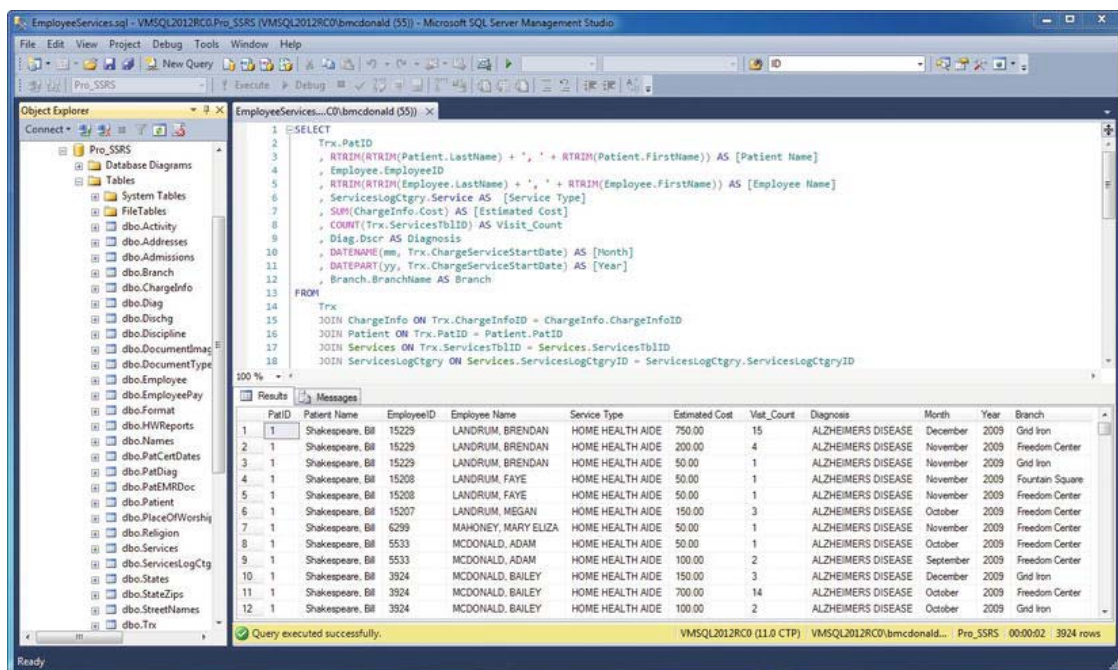


Figure 2-4. Viewing the query execution output in SSMS

The Execution Plan tab in SSMS shows graphically how the SQL query optimizer chose the most efficient method for executing the report, based on the different elements of the query. For example, the query optimizer may have chosen a clustered index instead of a table scan. Each execution step has an associated cost. Figure 2-5 shows the Execution Plan tab for this query.

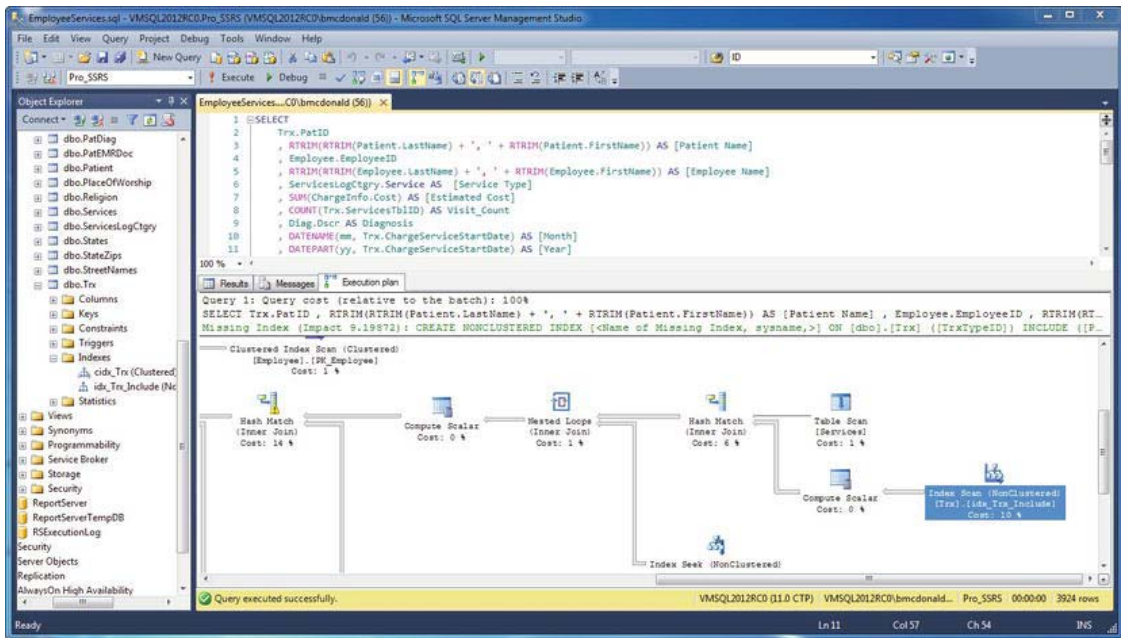


Figure 2-5. Viewing the Execution Plan tab displayed in SSMS

The query took one second to execute, and from this execution plan it's easy to see which section of the query had the highest cost percentage. There was a total cost of 11 percent when determining the TrxTypeID and the ServiceTypeID values used as a filter in the WHERE clause. For reference, the TrxTypeID integer field specifies the type of financial transactions as charges, payments, or adjustments. You're concerned only with the TrxTypeID value of 1, representing charges. For the service type, you're interested only in "V," representing visits, and not in other types of billable services, such as medical supplies. If you could get the cost of the WHERE clause down to a lower number, the query might improve the overall performance.

Optimizing Performance: Dividing the Load

Because SSRS and T-SQL share many data formatting and manipulation functions, you can choose in which process—query or report—these functions should be used. You can choose to have the query handle the bulk of the processing. This limits the number of rows that the report has to work with, making report rendering much faster. Alternatively, you can limit the selectivity of the query, allowing it to return more rows than are possibly required. You can then have the report perform additional filtering, grouping, and calculations, which allows the query or stored procedure to execute faster. With many users accessing the report simultaneously, having the report share the processing load also limits the impact on the data source server (in this case, SQL Server).

In this query, based on initial benchmarking, we've determined we'll remove the portion of the WHERE clause that specifies that the query should return only service types with a value of "V" for visits. Instead, we'll let the report filter out any service types that aren't visits. When you remove the service type criteria from the query and re-execute it, you can see that the overall execution time remains

constant at or is less than one second, and the cost of the `WHERE` clause goes from 11 percent to only 4 percent. Also, it's important to note in the performance analysis that the record count went up by only 35 records, from 3,924 to 3,959, by removing the "V" from the `WHERE` clause. You can see this in the lower-right corner of Figure 2-6.

To take advantage of a report filter, you need to add a field—`Services.ServiceTypeID`—to the `SELECT` and `GROUP BY` portions of the query, like so:

Select

...

Branch.BranchName AS Branch,
Services.ServiceTypeID

...

GROUP BY

...

Branch.BranchName,
Services.ServiceTypeID

You will use the additional field `Services.ServiceTypeID` as the filter value in the report that you will be designing. By proceeding in this fashion, even though you're returning more rows than you might need for a particular report, you also gain the benefit of using this same base query for other reports when you eventually make it a stored procedure, which you will do in the following sections. Other reports might need to show service types other than visits, and this query will serve this purpose with only slight modifications to the report. For example, you might need to investigate the cost or quantity of supplies (a service type of "S") used by employees. You can use this same query and stored procedure for that report. We could make use of a parameterized stored procedure that would allow various reports to pass in a value to be used to filter the results, but we'll progress to that in a moment.

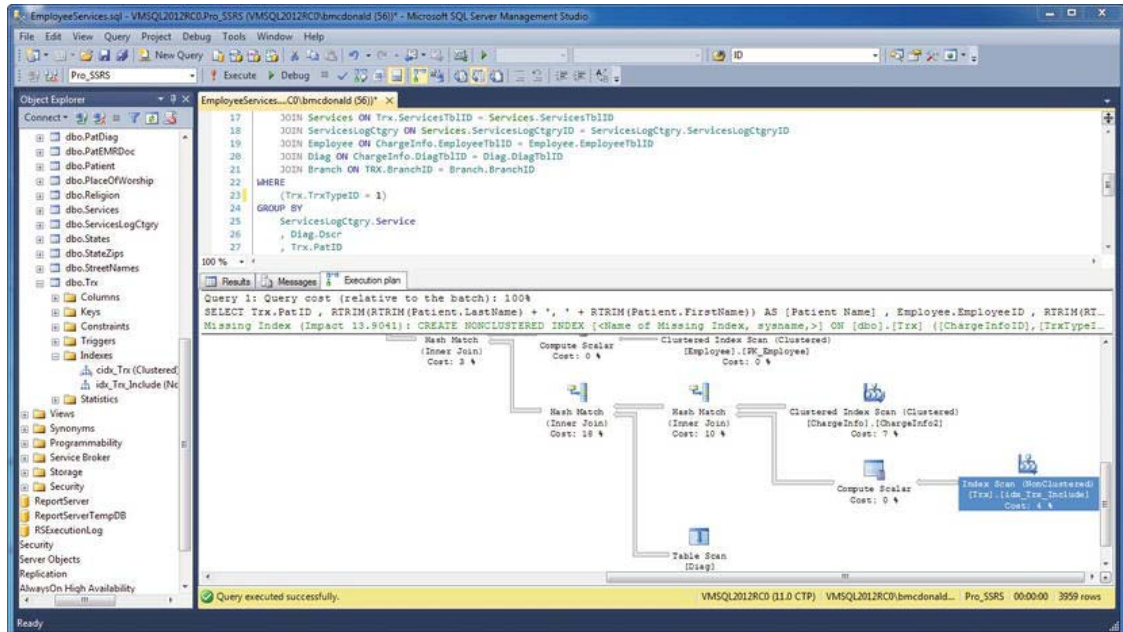


Figure 2-6. Viewing the execution plan with the modified query

The query as it stands, now including `ServiceTypeID` as a value in the `SELECT` clause and not as criteria, is ready to begin its life as a stored procedure. Queries serve many purposes, one of which is the development of reports, as you'll do in Chapter 6. However, encapsulating queries in stored procedures is typically the preferred method of deployment for several reasons. Stored procedures, like ad hoc queries, execute according to the execution plan generated by the query optimizer. Having the ability to reuse the execution plan saves time and resources. Stored procedures, which are also beneficial because they're precompiled, can reuse an execution plan even though the parameters passed to it at execution time might have changed values. You can hold stored procedures centrally on the SQL Server machine, unlike ad hoc queries that might be embedded in an application (or in the RDL file in this case). When the underlying schema of a database changes, you can update the stored procedure in one location, whereas embedded queries all need to be modified separately for each report in which they reside. In the next section, we'll show you how to create a stored procedure based on the employee cost query.

Using a Parameterized Stored Procedure

You can use SSMS to produce the code to create a stored procedure based on the employee cost query, and to drop it if the stored procedure already exists in the database. To create a stored procedure, expand the database where you want to create the stored procedure, navigate to the `Programmability` folder and expand it to see a folder named `Stored Procedures`. Right-click on that folder and choose, `New Stored Procedure`. This opens a window containing a sample `CREATE PROCEDURE` command for the new stored procedure.

```
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
```

To complete the new stored procedure, which you should name `Emp_Svc_Cost`, you simply need to paste in your `SELECT` statement. However, you can add optional parameters to limit the result set based on the following criteria:

- Service time (year and month)
- The branch where the employee works
- The individual employee
- The type of service

To create parameters for a stored procedure, you add the variable names, each prefixed by an `@` character and provide the appropriate data types along with their default values if desired. The default values for all the parameters are set to `NULL`, as Listing 2-4 shows. You can find the code for this query in the code download file `CreateEmpSvcCost.sql` in the `SQL Queries` folder.

Listing 2-4. Creating the Emp_Svc_Cost Stored Procedure

```
IF EXISTS (SELECT * FROM INFORMATION_SCHEMA.ROUTINES
           WHERE ROUTINE_NAME = 'Emp_Svc_Cost'
           AND ROUTINE_SCHEMA = 'dbo')
    DROP PROCEDURE dbo.Emp_Svc_Cost
GO
CREATE PROCEDURE [dbo].[Emp_Svc_Cost]
(
    @ServiceMonth INT = NULL
    , @ServiceYear INT = NULL
```

```

    , @BranchID INT = NULL
    , @EmployeeTblID INT = NULL
    , @ServicesLogCtgyID CHAR(5) = NULL
)
AS
SELECT
    T.PatID
    , RTRIM(RTRIM(P.LastName) + ' ' + RTRIM(P.FirstName)) AS [Patient Name]
    , B.BranchName
    , E.EmployeeID
    , RTRIM(RTRIM(E.LastName) + ' ' + RTRIM(E.FirstName)) AS [Employee Name]
    , E.EmployeeClassID
    , SLC.Service AS [Service Type]
    , SUM(CI.Cost) AS [Estimated Cost]
    , COUNT(T.ServicesTblID) AS Visit_Count
    , D.Dscr AS Diagnosis
    , DATENAME(mm, T.ChargeServiceStartDate) AS [Month]
    , DATEPART(yy, T.ChargeServiceStartDate) AS [Year]
    , S.ServiceTypeID
    , T.ChargeServiceStartDate
FROM
    Trx AS T
    INNER JOIN Branch AS B ON T.BranchID = B.BranchID
    INNER JOIN ChargeInfo AS CI ON T.ChargeInfoID = CI.ChargeInfoID
    INNER JOIN Patient AS P ON T.PatID = P.PatID
    INNER JOIN Services AS S ON T.ServicesTblID = S.ServicesTblID
    INNER JOIN ServicesLogCtgy AS SLC ON S.ServicesLogCtgyID = SLC.ServicesLogCtgyID
    INNER JOIN Employee AS E ON CI.EmployeeTblID = E.EmployeeTblID
    INNER JOIN Diag AS D ON CI.DiagTblID = D.DiagTblID
WHERE
    (T.TrxTypeID = 1)
    AND (ISNULL(B.BranchID,0) = ISNULL(@BranchID,ISNULL(B.BranchID,0)))
    AND (ISNULL(S.ServicesLogCtgyID,0) = ISNULL(@ServicesLogCtgyID,
        ISNULL(S.ServicesLogCtgyID,0)))
    AND (ISNULL(E.EmployeeTblID,0) = ISNULL(@EmployeeTblID,
        ISNULL(E.EmployeeTblID,0)))
    AND
    --Determine if Year and Month was passed in
    ((CAST(DATEPART(yy, T.ChargeServiceStartDate) AS INT) = @ServiceYear
        AND @ServiceYear IS NOT NULL)
        OR @ServiceYear IS NULL)
    AND
    ((CAST(DATEPART(mm, T.ChargeServiceStartDate) AS INT) = @ServiceMonth
        AND @ServiceMonth IS NOT NULL)
        OR @ServiceMonth IS NULL)
GROUP BY
    SLC.Service
    , D.Dscr
    , T.PatID
    , B.BranchName
    , RTRIM(RTRIM(P.LastName) + ' ' + RTRIM(P.FirstName))
    , RTRIM(RTRIM(E.LastName) + ' ' + RTRIM(E.FirstName))

```



```

        , E.EmployeeClassid
        , E.EmployeeID
        , DATENAME(mm, T.ChargeServiceStartDate)
        , DATEPART(yy, T.ChargeServiceStartDate)
        , S.ServiceTypeID
        , T.ChargeServiceStartDate
ORDER BY
        T.PatID
GO

```

Using ISNULL to Evaluate the Parameters

In the previous query, you added several new criteria to the WHERE clause for evaluating the parameters. One of those criteria was the ISNULL function, used to evaluate the values of the database fields and parameters.

```

(ISNULL(B.BranchID,0) = ISNULL(@BranchID,ISNULL(B.BranchID,0)))
AND
((CAST(DATEPART(yy, T.ChargeServiceStartDate) AS INT) = @ServiceYear
    AND @ServiceYear IS NOT NULL)
    OR @ServiceYear IS NULL)
AND
((CAST(DATEPART(mm, T.ChargeServiceStartDate) AS INT) = @ServiceMonth
    AND @ServiceMonth IS NOT NULL)
    OR @ServiceMonth IS NULL)

```

At first, the logic for these evaluations might seem a bit confusing, but remember that as long as the criteria are equal, results are returned. This is true through the entire WHERE clause because it's evaluated with AND. This is easier to understand with the following sample statement:

```
SELECT * from Table1 WHERE 1 = 1
```

In this statement, all rows are returned, because 1 always equals 1. It doesn't matter that you aren't comparing values from the table itself.

For the ISNULL function, you look to see whether the value of a database field—BranchID, for example—contains a NULL value, and if so, ISNULL replaces NULL with zero. The right side of that equation looks to see whether the @BranchID parameter was passed in as NULL; if so, then the value for @BranchID is set to the value of BranchID in the database table and equals every row. If the @BranchID parameter is passed to the stored procedure as a value—say, 2 for the branch Grid Iron—then only BranchID 2 is returned because BranchID = @BranchID = 2. This evaluation is performed when there might be NULL values in the field because NULL values can't be compared with standard operators such as =.

For the two time values, Service Year and Month, you use similar logic. If the parameters @ServiceMonth and @ServiceYear are passed in as NULL to the stored procedure, then the stored procedure returns every record. If the parameters contain legitimate values, such as 2009 for the year, the WHERE clause applies a filter when the parameter value equals the database value.

Query Performance and Indexes

While we're talking about designing efficient queries, there are many things that DBAs can do for a table to increase query performance. To list a few problems, statistics can get out of date, and indexes (where there are any on a table) can become fragmented or insufficient. Each of these will affect query

performance in different ways, but always in the wrong way—a query from a poorly designed or badly designed table will always take longer to execute than on from a table that is well designed and maintained.

You may not have appropriate permissions to manage indexes, but one thing you can do is to design a query to use the indexes that do exist. Here are a few tips to help you increase performance and decrease the time it takes to return results to your reports.

- *SELECT*: Only return the columns that are absolutely needed in your reports.
- *JOIN*: Only join tables that you need and use existing indexed columns in join conditions.
- *WHERE*: Use indexed columns in the order in which they are defined and eliminate the use of the LIKE operator using the wildcard (%) in front of the filtered value.

If your query is not performing as well as you would like, you may be able to ask your DBAs to run some performance checks on the table(s) that you are using in your query. They may just need to modify existing indexes to include the columns used in your queries. The cure could even be as simple as creating a maintenance plan on the database to update the statistics. Either way, try to use these tips when writing your queries. Performance-tuning queries is a large subject and many books are available to help you write optimized queries, such as *SQL Server 2012 Query Performance Tuning* by Grant Fritchey (Apress 2012).

Column and Table Aliasing

Column and table aliasing does not actually make your queries run more efficiently. However, it does make them easier to read and quicker to write. Aliasing lets you use an abbreviated (or more descriptive) label for columns and tables. This way, you don't have to type out the entire name of a table every time you use a column from it, or you can assign a more appropriate name for a column. Using the AS keyword, we tell SQL Server to alias the field or table as some other label.

```
SELECT
```

```
...
, DATENAME(mm, T.ChargeServiceStartDate) AS [Month]
```

```
...
INNER JOIN ServicesLogCtgrY AS SLC ON S.ServicesLogCtgrYID = SLC.ServicesLogCtgrYID
```

This statement uses the DATENAME function to alias the Trx table's ChargeServiceStartDate as the Month. The next line shows how you can alias a table named ServicesLogCtgrY so you can refer to it as SLC. This way, whenever you need a column from the ServicesLogCtgrY table, you can reference the alias followed by a period, then the column name.

Testing the Procedure

The next step is to grant execute privileges for the stored procedure in SSMS by navigating to the database Pro_SSRS and then expanding the Programmability folder. From here, select Stored Procedures, right-click Emp_Svc_Cost, and finally select Properties. A Permissions property page will allow you to add the public role and grant execute permission to any group or user you desire. In this case, click Add on the Permissions page, find Public in the list of available users, and grant the Execute permission. (We're sure the humor of this wasn't lost on the developer, who knew someone would grant a public execution.)

■ **Note** The test server on which we’re developing the reports is an isolated and secure system. Typically, granting execution privileges to the public role isn’t recommended. We’ll lock down both the stored procedure and the report in Chapter 11.

You can now test the procedure directly in SSMS with the following command:
EXEC Emp_Svc_Cost

Because you have allowed NULL values for the parameters, you don’t explicitly have to pass them in on the command line. However, to test the functionality of the stored procedure, you can pass in the full command line with the appropriate parameters; for example, you can pass all services rendered in September 2009, like so:
EXEC Emp_Svc_Cost 09, 2009, NULL, NULL, NULL

Executing the procedure in this way returns 893 records in a fraction of a second, and the results verify that, indeed, only services in September 2009 were returned (see Figure 2-7).

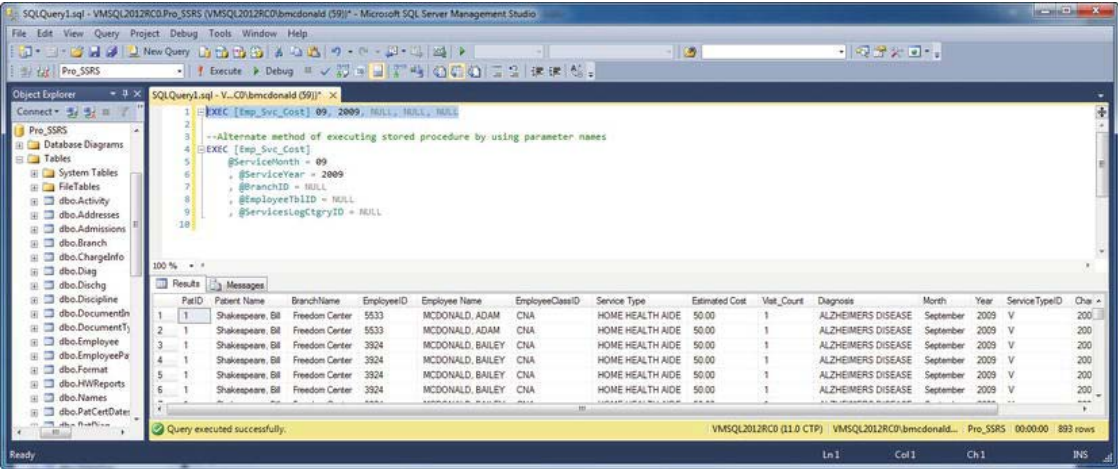


Figure 2-7. Viewing the results of Emp_Svc_Cost with year and date parameters

Summary

In this chapter, you began to design the essential part of a report: the query and stored procedure. By using stored procedures, you gain the benefits of central administration and security while also gaining the ability to execute compiled code to return the dataset instead of a stand-alone query. You can develop queries in conjunction with the report, using the built-in query tools within SSRS. However, it is considered best practice to deploy the report with a stored procedure whenever possible.

A report request and the target audience are the deciding factors when determining the layout and default rendering of the report. However, even though reports are often designed to answer a specific

need, if they're based on the same tried-and-true stored procedures, with similar parameters and groupings, the data will be accurate across all reports. You can then focus the design time on the report itself and not on rewriting queries.

Introduction to Reporting Services Design with SQL Server Data Tools

The professional lines separating system administrators, DBAs, and developers are blurring. Products are often extensible through code or at least have the potential to create functionality that goes well beyond that of out-of-the-box offerings. SSRS is such an application. The days of the Microsoft Management Console (MMC) are numbered and will be overshadowed by the new interface on the block, the IDE, though even that isn't new by any means, as any developer will tell you. However, system administrators, DBAs, and even report designers have had to become familiar with this new way of performing their day-to-day tasks. As you're probably well aware, you can create reports for SSRS within Visual Studio 2005 and up, or within BIDS/SSDT. To remind you of these abbreviations, the designer included with SSRS 2005 through the 2008 R2 release was labeled as Business Intelligence Development Studio (BIDS). However, in the 2012 release, Microsoft decided to re-label the designer as SQL Server Data Tools (SSDT) because of the inclusion of features such as further development integration with SQL Azure environments. The ability to use SSDT/BIDS is advantageous for Visual Studio developers, because now they can use the same IDE for report creation and application development! For the rest of us, creating reports in Visual Studio 2010 presents a learning curve.

SQL Server 2005 introduced two new management and development environments, BIDS, a subset of Visual Studio 2005, and SSMS. In SQL Server 2008, 2008 R2 and 2012, these applications have been enhanced to improve the integration of design and management functionality in a common set of tools.

Now that you have developed your queries and stored procedures, you can turn your attention to the tools available to report designers when SQL Server is installed. Over the next three chapters we'll familiarize you with the tools of report design before showing you how to create a full-blown SSRS report in chapter 6. The topics we'll cover include the following:

- The elements of SQL Server Data Tools (SSDT/BIDS)
- The role of RDL in SSRS with sample code from the various report objects it controls
- Creating a data source and dataset
- Defining query and report parameters
- Discussing report pagination
- Defining expressions and filters to demonstrate how you can use them together to control report content and formatting
- Implementing Tablix properties, new in SSRS 2008, which extend the functionality of several data regions by combining row grouping and column grouping

- Creating data region samples: the List, Textbox, Table, Rectangle, Matrix, Chart, and Image
- Implementing Maps, Data Bars, Sparklines and Indicators, all of which provide more sophisticated dashboard style elements to reporting

In this chapter, we'll show you how to set up and explore the SSDT/BIDS IDE using the embedded elements of SSRS within the 2010 environment.

Everyone learns differently—some like to follow a step-by-step guide to a known conclusion, and some like to view a completed report to see the specific components of its design. We will therefore offer both approaches in this and subsequent chapters. Specifically, we'll show you how to build each sample from the ground up and we'll also point you to the completed sample in the Source Code download available from this book's catalog page on www.apress.com, so you can analyze the report as you work through the steps to achieve the end result.

We provide all the data sources, reports, and projects you will work with throughout this chapter in a solution called `Pro_SSRS`. You can open this solution in both BIDS and Visual Studio 2010. As we mentioned earlier, Microsoft has changed the label from Business Intelligence Development Studio to SQL Server Data Tools (SSDT), but we will use BIDS and SSDT interchangeably throughout this book. You'll find detailed instructions in the Source Code/Download area on the Apress site for installing the samples for each chapter. This chapter will focus primarily on the IDE of BIDS and provide a step-by-step guide to familiarize you with BIDS. Chapter 4 will describe the use of SSRS-specific report objects such as Lists and Tables, and Chapter 5 will introduce report objects like Charts, Gauges, Maps, Sparklines, and Data Bars. When you begin creating these specific report examples, you will have two reports for each object, a starting-point report and a completed report sample; we will point these out at the beginning and end of each main section. This approach lets you step through the procedure to produce the output in the starting-point report and then open and compare the end result to the completed report.

Exploring the Elements of BIDS

In BIDS, one or more *projects* contain all the reports and shared data sources. A project physically and logically groups reports together, and maintains properties specific to that project. These properties allow the project to work independently of other projects. All created projects are themselves contained within a *solution*. A solution is simply a collection of one or more projects that are made available to Visual Studio. A single solution can contain a Reporting Services, an Integration Services and an Analysis Services project as well as a Windows or Web application project.

We'll now show you how to fire up BIDS to create a solution and a Reporting Services project. To create a solution, you'll need to have BIDS loaded. Navigate to the shortcut to load the `devenv` executable by choosing **Start** ► **All Programs** ► **Microsoft SQL Server 2012** ► **SQL Server Data Tools**. Both SQL Server Data Tools (SSDT) and Business Intelligence Development Studio (BIDS) are shortcuts to the `devenv` executable. Throughout this chapter and the rest of the book, we may refer to SSDT as BIDS because most development initiatives using this tool are Business Intelligence related. However, BIDS and SSDT are interchangeable terms. Once you have BIDS open, you can get to the new project screen by selecting **File**, and then **Project** under the **New** submenu of the menu bar or simply by clicking the **New Project** button on the **Start Page** as shown in Figure 3-1.

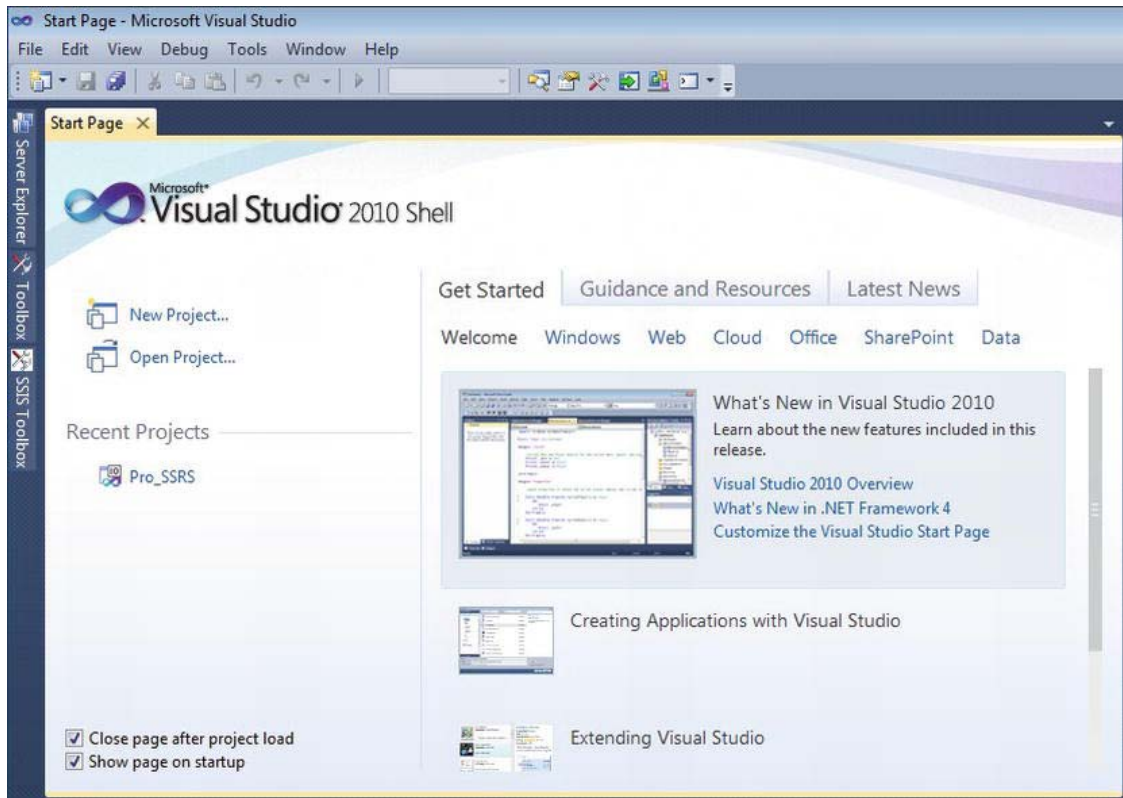


Figure 3-1. Business Intelligence Development Studio Start Page

The New Project dialog box opens. Select Report Server Project under Installed Templates ► Business Intelligence ► Reporting Services. If this is the first project that you have created, the project name defaults to Report Project1. The location of the project can be a local drive or a network location. In this case, make it C:\Pro_SSRS\Reports.

By default, the solution is named according to the project name, Report Project1, as shown in Figure 3-2. If you check the Create Directory for Solution box, you can append a new directory to the base location. In this case, choose Solution1. After you click OK, both the project and solution are created, and you can create report items and data sources within the project.

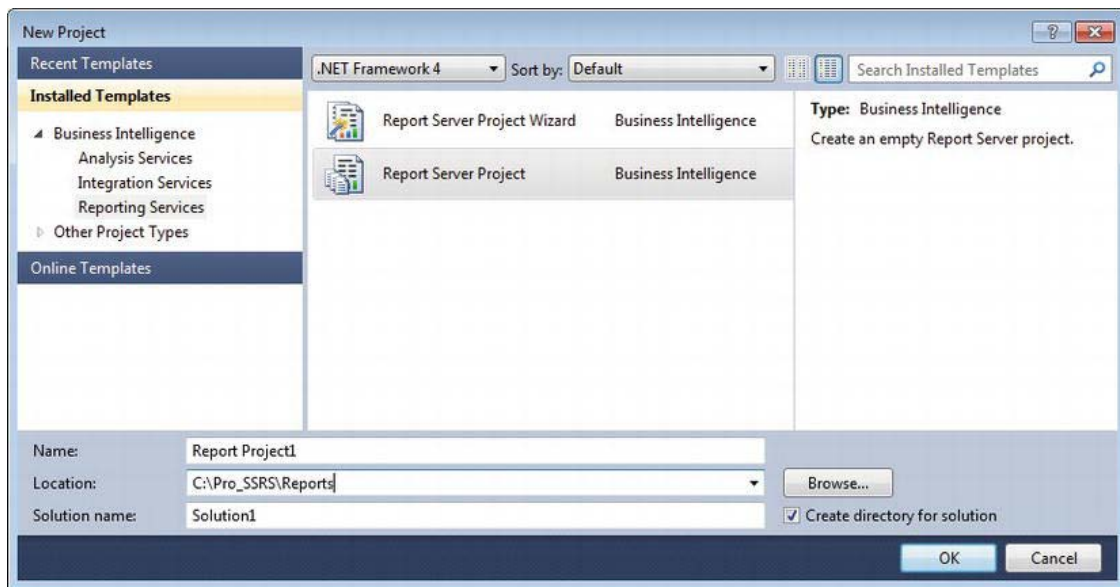


Figure 3-2. *New Project dialog box*

If you know where your reports are going to be deployed, it is considered good practice to add a few important project property settings: the target folder in which to deploy the reports on the SSRS server, and the SSRS server URL. You can view and set these properties using the Solution Explorer, whose window displays the opened solution and the projects it contains, as well as all the reports and other objects the individual projects might contain. By default, BIDS locates the Solution Explorer on the right side of the environment, but it may be docked or floating in a different location. If you do not see it, click **View ► Solution Explorer** on the menu bar. Highlight the project in the Solution Explorer and choose **Project ► Properties** from the menu (or right-click the project and select **Properties**) to see a window like the one in Figure 3-3. The `TargetReportFolder` property controls the folder that is created to store the deployed reports and data sources on the SSRS server. You can use the `TargetDataSourceFolder` and `TargetDatasetFolder` properties to store the project specific data sources and shared datasets. The `TargetServerURL` property is the SSRS Web server URL (when you start to use SharePoint Integration Mode, as you will when you get to Chapter 12, this property will contain the SharePoint server URL). As Figure 3-3 shows, the `TargetServerURL` property is in the form `http://servername/ReportServer`. In this case, the SSRS Web server is `localhost`.

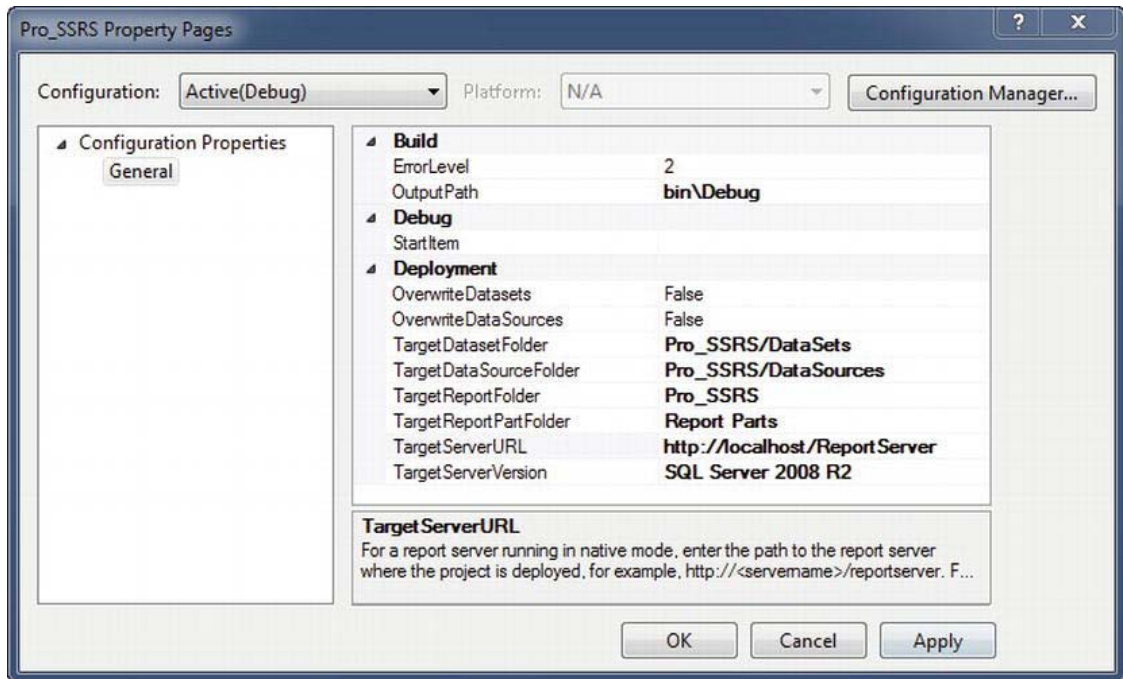


Figure 3-3. Project properties

■ **Note** We'll cover some of these other settings, such as `TargetDatasetFolder`, `TargetDataSourceFolder`, and `TargetReportPartFolder`, in Chapter 8. However, unless the object needs to be consumed by report developers or end users with no access to the specified `TargetReportFolder`, it's good practice to store your project objects using the folder structure that you set in the `TargetReportFolder`. For example, the `TargetReportFolder` in Figure 3-3 is `Pro_SSRS` and the `TargetDataSourceFolder` is set to `Pro_SSRS/DataSources`. This prevents other developers overwriting your data sources, datasets or report parts.

Setting Up a Basic IDE

Now that you have a new solution and a new project to contain the reports you'll build, it's time to get personal. As a report designer, you're going to spend many hours gazing at the pixels that are your creations, so it's important to set up the environment exactly the way you want it. The ideal setup for designing reports is a personal choice. Some prefer high-resolution display settings with every available design toolbar always in view within the environment, while others like undocked toolbars and a dual monitor set up at a lower resolution. Whatever your preference, BIDS makes it easy to manipulate the design tools within the IDE to personalize your configuration. In addition to the Solution Explorer,

covered in the previous section, you can use several common tools within the IDE to design reports:

The Toolbox: This is where you'll find all the report objects covered in this chapter, such as the Matrix and Table data regions. Data regions are the defined report objects within the SSRS report design environment that contain the field values from the data set.

The Properties window: Here, you set the values for the various formatting and grouping properties for report items.

The Error List window: You will need this when troubleshooting report errors. Mismatched data types and invalid use of functions are common issues that arise when designing reports. The Error List window is the place to see the details of these errors.

The Report Data window: This holds the built-in fields, data sources, datasets, images, and the field information you've defined for the report.

Figure 3-4 shows a custom layout for designing reports in SSRS. You can dock any toolbar anywhere within the IDE or position them beyond it on the desktop. This setup works best with a high-resolution configuration, 1152 × 864 or higher. In the setup in Figure 3-4, the Toolbox is undocked, and the Solution Explorer is on the right side of the report design grid. If you wish, you can set the toolbars to hide automatically when not in use—in this example, the button to display the hidden Report Data pane is shown at top left. BIDS, like the full Visual Studio 2010 environment, has a dock position map that assists in precisely placing the dockable item. You can see the map, with a slight transparency at all edges and in the middle, as you move items around within the report designer.

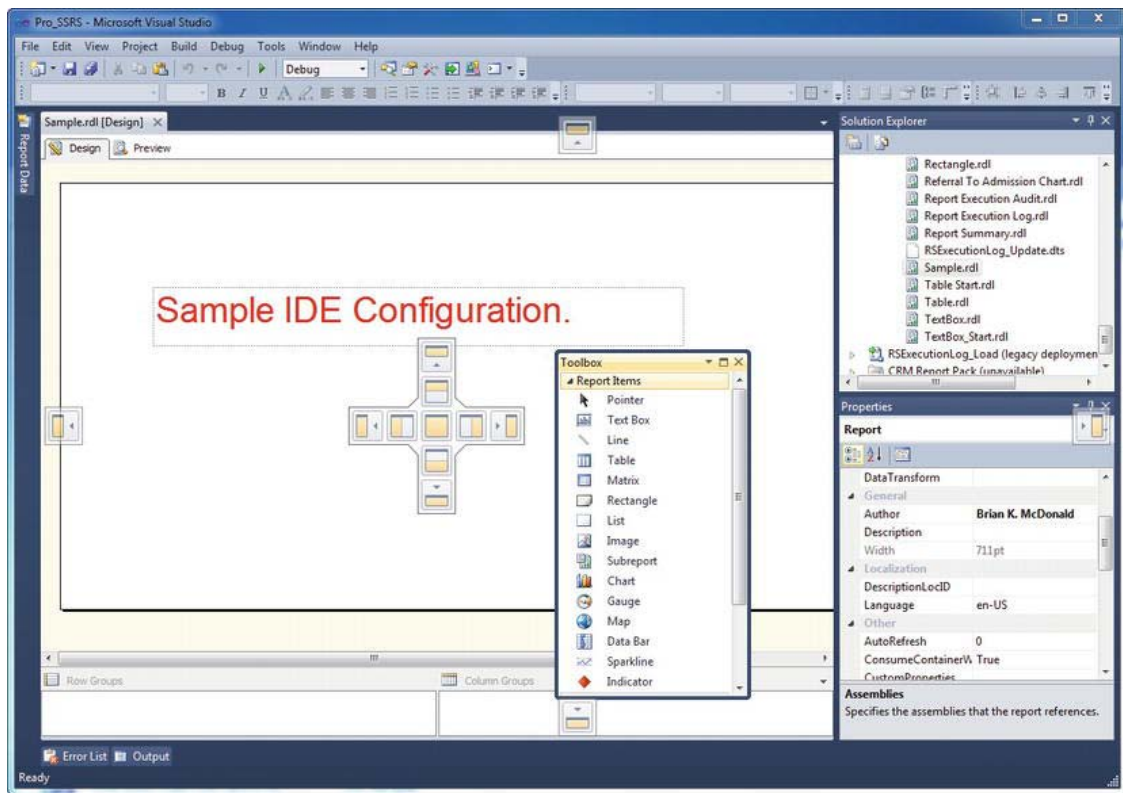


Figure 3-4. Sample IDE configuration

Understanding Report Definition Language (RDL)

RDL is the standard to which all reports created with the embedded SSRS tools in Visual Studio 2010 and other SSRS services adhere. RDL is an XML-based schema that defines each element of a report, such as formatting, dataset information, grouping and sorting, and parameters and filters. As you add items to the report, the RDL code changes to include each addition.

In the IDE, these code changes are typically invisible, as they take place in the background. Occasionally, however, you might need to modify the RDL directly to make global changes using a find-and-replace method. We've had to do this several times when a parameter or field name changed in a source's stored procedure. When you are working on a report in Visual Studio 2010, you can view the RDL code directly by clicking **View** ► **Code** from the menu or by right-clicking a report and selecting **View Code**.

Visual Studio 2010 and BIDS are the main report designers for SSRS currently, but as more and more companies embrace RDL, other report designers may become available. In fact, in SQL Server 2005, Microsoft introduced a Report Builder application that allows an end user to design and publish SSRS reports. SQL Server 2008 included a new version, appropriately named Report Builder 2.0, with Microsoft Office Ribbon technology and a complete facelift taking it one step further toward putting

report development into the hands of report requesters. Report Builder 3.0, released with SQL Server 2008 R2, had even more features for ad hoc end-user reporting. Report Builder 1.0 and 2.0 are still available for backward compatibility, and 1.0 still relies on report models for its data sources, unlike 2.0 and 3.0. Chapter 13 covers Report Builder 1.0, 2.0, and 3.0 applications, and their components, in more detail.

Throughout this chapter, we'll present the RDL sections of the report objects on which you are working to show how the RDL is updated while designing a report. The complete RDL schema is available at <http://schemas.microsoft.com/sqlserver/reporting/2008/01/reportdefinition>.

Adding a Report

In line with Microsoft's general policy, there is more than one way to add a report to a project. One way is to employ a wizard to work through the report creation process, but for now, we are just going to add a blank report to our project. Right-click the Reports folder in the Solution Explorer, select Add, and then New Item. Notice that you have the option of adding an existing item as well. This option is useful if you already have a report to add to a project or if you've built a template report file as a base starting point. For now, select Report in the Add New Item dialog box as shown in Figure 3-5 below, and then click Add to create a blank report named Report1.rdl in the project.

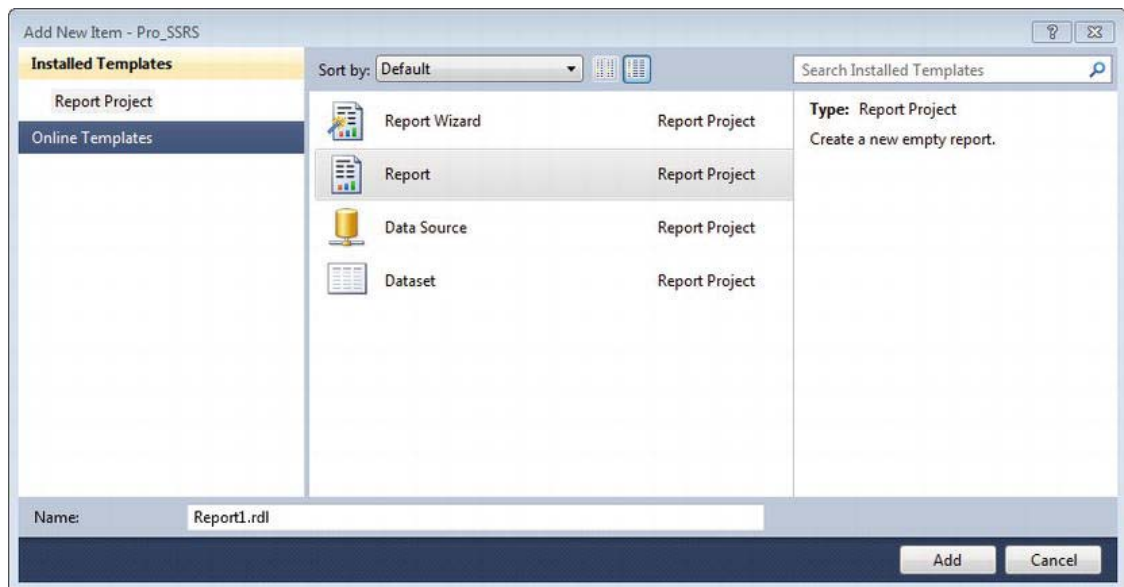


Figure 3-5. Add New Item Dialog Box in BIDS

The new report should open up in the design environment, but if it doesn't, double-click the new report in the Solution Explorer. By default the report is named Report##.rdl, where ## is the next available report number in sequence. At this point, the report is a blank slate. Figure 3-6 shows the IDE, including the Solution Explorer, Toolbox and Report Data boxes. If you are familiar with creating reports in Visual Studio 2005, you will notice that the Data tab has been separated from the Design and Preview tab and now resides in a dedicated Report Data window. In the SQL Server 2008 R2 release, we were

provided another folder in the Report Data pane called Data Sources. For now, just keep in mind that this new IDE has been designed to centralize your local and shared data sources. Again, coming from VS 2005, you will notice the new Row Groups and Column Groups areas, designed for easier management of Tablix-style groupings. As with any report that contains data from a data source, the first step is to create a link between the report and the data source as well as one or more datasets. We will do both in the next section.

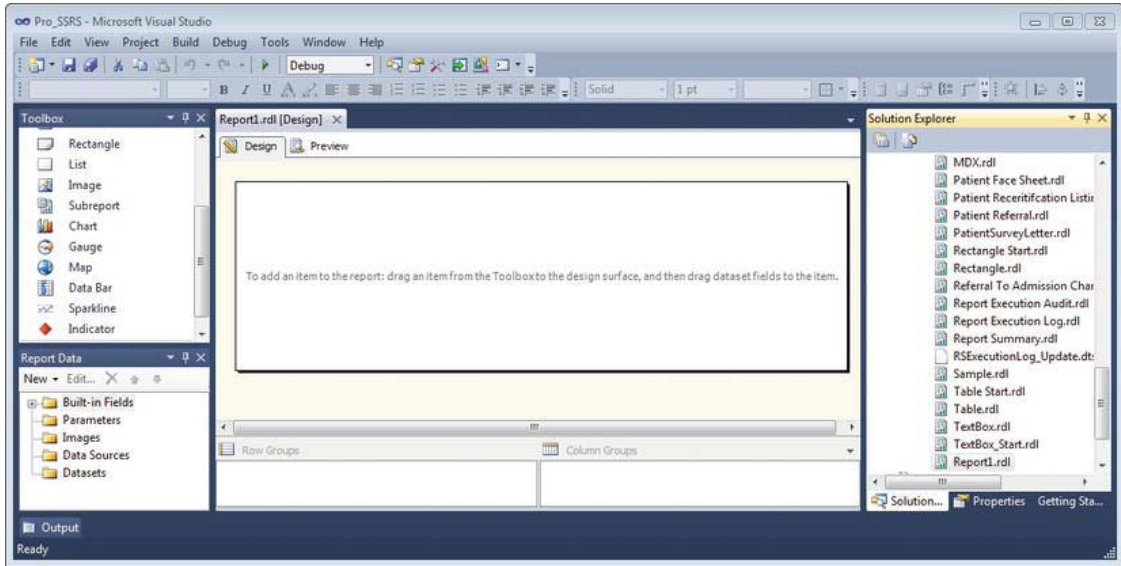


Figure 3-6. BIDS IDE with Report1.rdl loaded

Setting Up Data Sources and Datasets

Each report created in SSRS contains both a *data source* and a *dataset*. A data source not only defines the type of connection that is established to retrieve the data, whether it's SQL Server, Analysis Services, or Oracle, but also defines the specific connection properties, such as the server, database name and security credentials. A dataset, on the other hand, is the data, rows and columns, or fields that are returned from the data source. The dataset is created by building a query that retrieves information from the data source. This query, as in the case of a SQL Server data source, can be a text-based query embedded directly in the report, or a stored procedure.

In Chapter 2, you created a stored procedure called `Emp_Svc_Cost`, containing employee and patient visit information. You'll use that procedure as your dataset as you work with most of the report objects in this chapter. For other report objects, such as the image report object, you will use direct queries instead of the `Emp_Svc_Cost` stored procedure.

Creating a Data Source

Each report can use one or more data sources. Reports that use the same data source—for example, one that connects to a specific SQL Server database—can use what is referred to in SSRS as a *shared data source*. Shared data sources are published along with the report and can be modified on the report server

after deployment. In BIDS, shared data sources contain several properties that you must configure before you can use them.

Let's step through the process of creating the shared data source for the stored procedure `Emp_Svc_Cost`. First, right-click Shared Data Sources in the Solution Explorer, and select Add New Data Source; the Shared Data Source Properties dialog box will appear. Second, click the Edit button in the dialog box to create the connection string. In this case, you know that the server that contains your source database and stored procedure is located on the local SQL Server, so you can type `localhost` as the server name. Because we left the default connection type on the previous window, the data source property is set to Microsoft SQL Server (SqlClient). After you type `localhost` as the server, you can choose the `Pro_SSRS` database from the database drop-down selection. In this case, because the database is configured to use both Windows and SQL authentication, choose the Use Windows Authentication option. If you choose to use SQL authentication, you can also choose to store the SQL username and password. Generally, Windows Authentication is the preferred method because it has a single point of login for users (Chapter 11 covers authentication for deployed reports). Figure 3-7 shows the data source connection properties. You can test the connection by clicking the Test Connection button.

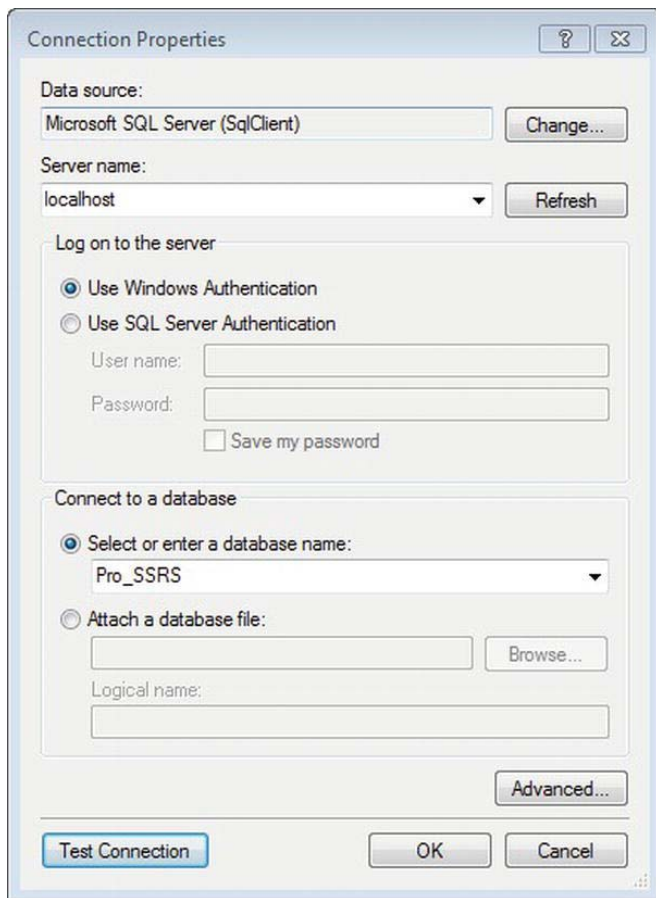


Figure 3-7. Data source connection properties

■ **Note** In Reporting Services 2012, you can choose from many types of data source. SQL Server happens to be the default, but you can grab data from sources such as SQL Azure, Analysis Services, Oracle, SAP, and TERADATA. You can use generic OLE DB and ODBC data sources as well, of course.

You now have a shared data source. Notice that its name has defaulted to `DataSource1`. You can rename it by right-clicking it, selecting `Rename`, and entering `Pro_SSRS.rds` as the new name. It is good practice to give each data source a meaningful name, such as the name of the database it will connect to. In a real-world environment, you will be deploying your reports and/or databases on different servers, so it's not advisable to prefix the data source name with the name of the server containing the database. In other words, it is not recommended to name your data source `DEV_Pro_SSRS` to represent a development data source, would you want to name your data source something like `Server1_Pro_SSRS`.

In practice, we've developed all our reports using an identical data source name. However, because each of our online customers had a database name that uniquely identified them, we designed an application that reset the database properties in the data source after it was published. In this way, we could use the same reports against the same database schema, but we could deploy them to multiple customers on the same report server.

In this example, the data source file you created has an `.rds` extension and is stored and published separately from the report. You can open an `.rds` file in a text editor, because it's an XML file that defines the connection properties you just created graphically. Listing 3-1 shows the `Pro_SSRS.rds` file.

Listing 3-1. Pro_SSRS.rds File

```
<?xml version="1.0" encoding="utf-8"?>
<RptDataSource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Name>Pro_SSRS</Name>
  <ConnectionProperties>
    <Extension>SQL</Extension>
    <ConnectionString>Data Source=localhost;Initial Catalog=Pro_SSRS </ConnectionString>
    <IntegratedSecurity>true</IntegratedSecurity>
  </ConnectionProperties>
  <DataSourceID xmlns="http://schemas.microsoft.com/SQLServer/reporting/
    reportdesigner">99401d15-cd9b-489a-b3ee-e027de26a4e0 </DataSourceID> </RptDataSource>
```

Creating a Dataset

Your next step is to proceed to the Report Data area to create your first dataset. This is true regardless of whether you've developed a query or stored procedure in an application other than BIDS or whether you're beginning it now within the report. In this example, you are going to use a stored procedure that is already complete and tested, so half the battle is done.

Using BIDS to develop SSRS 2008 R2 and SSRS 2011 reports, you create a dataset in two steps: creating a link to a data source and getting the data for the report.

1. In the Report Data box, select `<New ~Dataset...>`, which opens the Dataset Properties dialog box. Each dataset defaults to a name of the form `DataSet##`, where `##` is the next unused number in sequence.

2. You can use a shared dataset but in this example you will utilize a dataset embedded in the report. Choose Use a dataset embedded in my report option.
3. Now create a link between this dataset and the data source that we created earlier. Click the New button next to the data source drop-down box.
4. In the Data Source Properties window, name the data source Pro_SSRS and choose *Use shared data source reference*.
5. Click on the drop-down list box and select the Pro_SSRS shared data source that we created earlier.
6. Figure 3-8 shows the Data Source Properties windows with these settings. Click OK to save your settings.

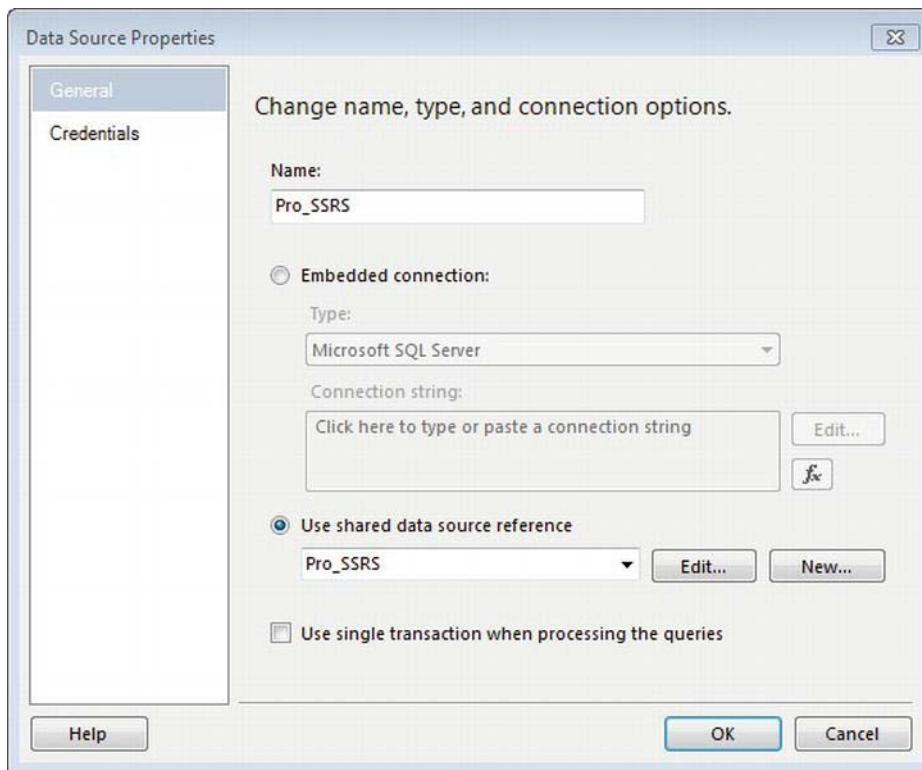


Figure 3-8. Data source properties

Now that we have a link to our shared data source, we can point the dataset to our stored procedure. In the Query Type section, change the default, Text, to Stored Procedure. In the Select or enter stored procedure name textbox, input the name of your stored procedure, Emp_Svc_Cost, as shown in Figure 3-9. Finally, enter Pro_SSRS_DS for the name of the new dataset and click OK to complete the dataset configuration.

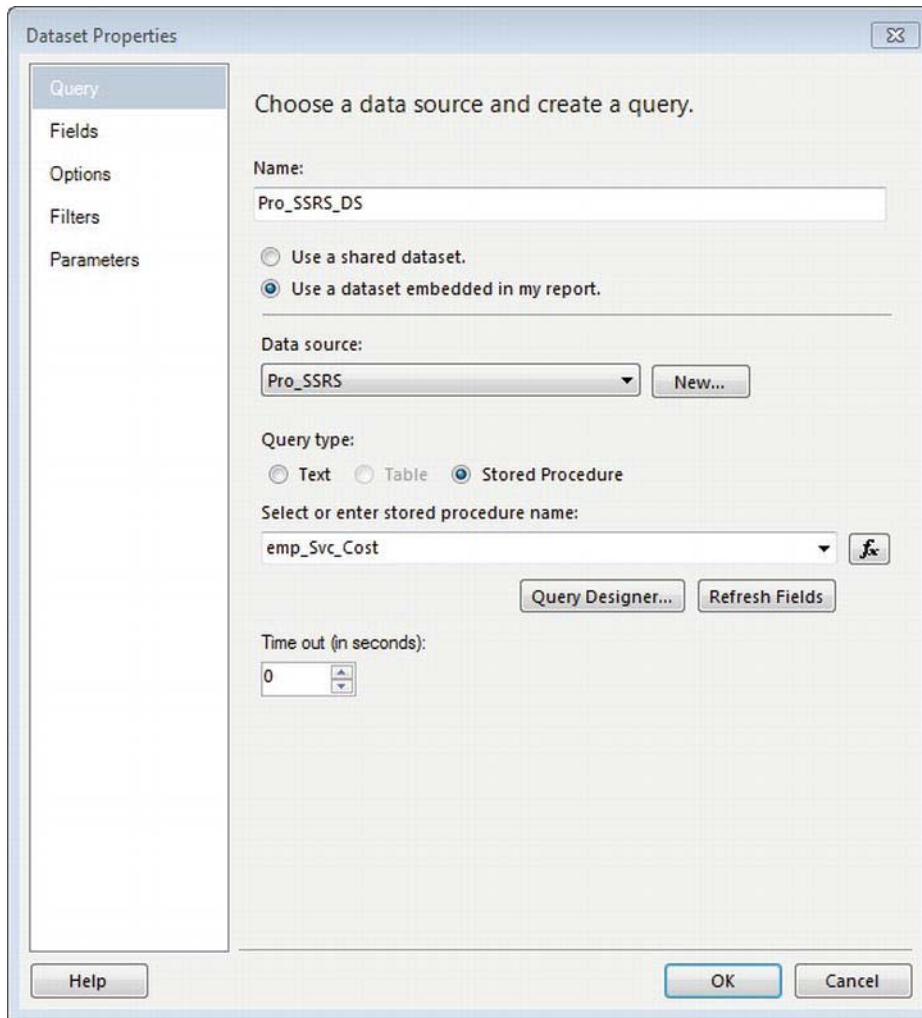


Figure 3-9. Dataset properties

When you click OK, a few things happen behind the scenes in the Report Data window. A data source called Pro_SSRS is created under the Data Sources folder and a dataset named Pro_SSRS_DS is created under the Datasets folder. This dataset is populated with the available fields from the stored procedure. At the same time, BIDS creates all of the parameters that the stored procedure accepts. You can't see the actual data at this point, but you can right-click the dataset and select Query... to bring up the default generic query editor where you can execute the stored procedure. At execution time, you are prompted for any parameters that have been defined by the stored procedure; you must supply the parameter values before data is returned. In the case of this stored procedure, define five parameters: @ServiceYear, @ServiceMonth, @BranchID, @EmployeeTblID, and @ServicesLogCtgyID. The available default values for the parameters, when the stored procedure is executed in the Report Data box, are

either NULL or Blank. Unlike a NULL value, a Blank value can be an empty string. A NULL value indicates that the value is unknown, and NULL values can't be evaluated with non-NULL values. In Chapter 2, you built logic into the stored procedure to handle NULL parameter values so that when the user doesn't supply a value, the query returns all records. If the user selects a specific value, only the records that match that parameter value are returned. Execute the stored procedure with NULL values, and make sure you're getting the results you expect. You can see the results of the stored procedure execution in Figure 3-10.

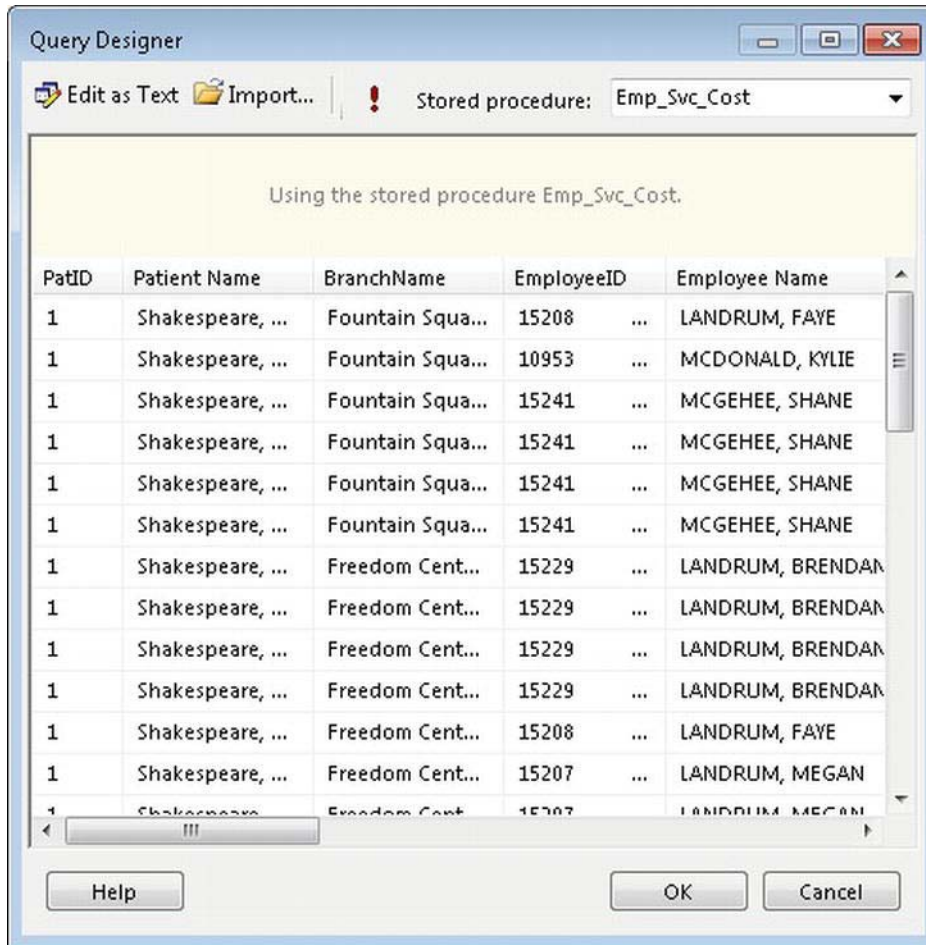


Figure 3-10. Data returned from the stored procedure

■ **Note** When you are prompted for the parameter values, Blank is the default value. To have the stored procedure execute without a data type error, you must select the value of NULL.

An SSRS report can use multiple datasets simultaneously. This extends the flexibility of your reporting, as you can provide more data to the user in a single report. Multiple datasets are also useful for populating parameter drop-down lists, which you will do in Chapter 6. However, having too many datasets could affect a report's performance, so it's important to make sure the execution times for each result set are within acceptable ranges.

The RDL file for each report contains a section for each dataset defined for the report. Listing 3-2 shows a sample of the RDL for the dataset you've defined in this chapter.

Listing 3-2. Dataset Section of RDL

```
<DataSets>
  <DataSet Name="Pro_SSRS_DS">
<Fields>
  <Field Name="PatID">
<DataField>PatID</DataField>
  <rd:TypeName>System.Int32</rd:TypeName>
  </Field>
  <Field Name="Patient_Name">
<DataField>Patient Name</DataField>
  <rd:TypeName>System.String</rd:TypeName>
  </Field>
  <Field Name="BranchName">
<DataField>BranchName</DataField>
  <rd:TypeName>System.String</rd:TypeName>
  </Field>
  <Field Name="EmployeeID">
<DataField>EmployeeID</DataField>
  <rd:TypeName>System.String</rd:TypeName>
  </Field>
  <Field Name="Employee_Name">
<DataField>Employee Name</DataField>
  <rd:TypeName>System.String</rd:TypeName>
  </Field>
  <Field Name="EmployeeClassID">
<DataField>EmployeeClassID</DataField>
  <rd:TypeName>System.String</rd:TypeName>
  </Field>
  <Field Name="Service_Type">
```

When creating a dataset, several additional tabs contain other configuration properties:

Fields: Defines additional fields such as calculated fields or fields that aren't automatically defined with the data source. You derive calculated fields from an expression.

Options: Sets several options specific to the data as it's retrieved from the data provider, such as case sensitivity and collation.

Parameters: Defines the query parameter values for the dataset and the order in which they're evaluated. Stored procedures with declared parameters automatically generate the query parameters in SSRS.

Filters: Defines filter values for the dataset that you can use when the report is executed.

Creating Other Data Sources

One exciting aspect of SSRS is its ability to query multiple data source types as well as SQL Server. As mentioned previously, any ODBC or OLE DB provider can be a data source for SSRS, as can XML, SSIS, and SAP. For a simple example of using a data source other than a SQL Server database, let's look at the OLE DB Provider for Microsoft Directory Services. Creating the data source to Directory Services is similar to procedure you used to create the SQL Server data source, except that you select OLE DB as the data source and OLE DB Provider for Microsoft Directory Services for the OLE DB provider in the data source properties.

By using a direct LDAP query, you can generate field information for use in SSRS like so:

```
SELECT en,sn,objectcategory,department
FROM 'LDAP://DirectoryServerName/OU=OuName,DC=Company,DC=Com'
```

The query uses a standard SQL dialect that returns the common name, surname, object category (computer or person), and department from Active Directory. The field names are automatically created and can be used like any other data field for a report.

You must consider a couple of caveats when querying Active Directory or any other data source that doesn't support the graphical query designer in SSRS:

- Query parameters aren't supported directly in the query. However, you can define and use report parameters in the query—referred to as a *dynamic query*—and to filter data.
- Because a graphical query designer isn't available, you need to develop the query in the generic query designer by typing the query directly and testing. This requires knowledge of Active Directory objects and names.

■ **Tip** Several tools are available to assist in managing Active Directory, such as Active Directory Application Mode (ADAM); LDP, an Active Directory tool; and ADSIEdit, a graphical Active Directory browser. Both are included with the Windows Support Tools.

Configuring Parameters

Parameters in SSRS come in two flavors, *query parameters* and *report parameters*, and the two are often tied together closely.

You use a parameter that's based on a SQL query or stored procedure to limit the record set returned to the report, typically in the WHERE clause of a query. In the source query, you define parameters by prefacing the parameter's name with an @ symbol, such as @MyParameter. Within SSRS's query design tools, this does two things: it forces the query to prompt for the value of the parameter when it's executed. Secondly, it automatically creates the other parameter, the report parameter, using the same name. With stored procedures, such as Emp_Svc_Cost, which you created in the previous chapter and have used here, any parameters that have been defined in the stored procedure are also automatically created for the report.

Report parameters can exist that are disassociated from a query or stored procedure. For example, you could have a report parameter that controls a report's behavior or layout properties. When you use a report parameter in this way, it's often linked to a report filter or used in an expression that controls a property value of a report item. In Figure 3-11, you can see the report parameters that were created automatically for us when we executed the `Emp_Svc_Cost` stored procedure. You'll also see the Report Parameters dialog of the individual parameters, such as `ServiceMonth`, which you open by either double-clicking the parameter or right-clicking and selecting `Parameter Properties`. The report parameters, in Visual Studio or BIDS, now appear consolidated in the Report Data box, whereas in previous versions they were contained within their own report parameters property box. Report parameters are used within a report, both for setting criteria for datasets and for controlling report design layout elements, which you will do in detail in Chapter 6.

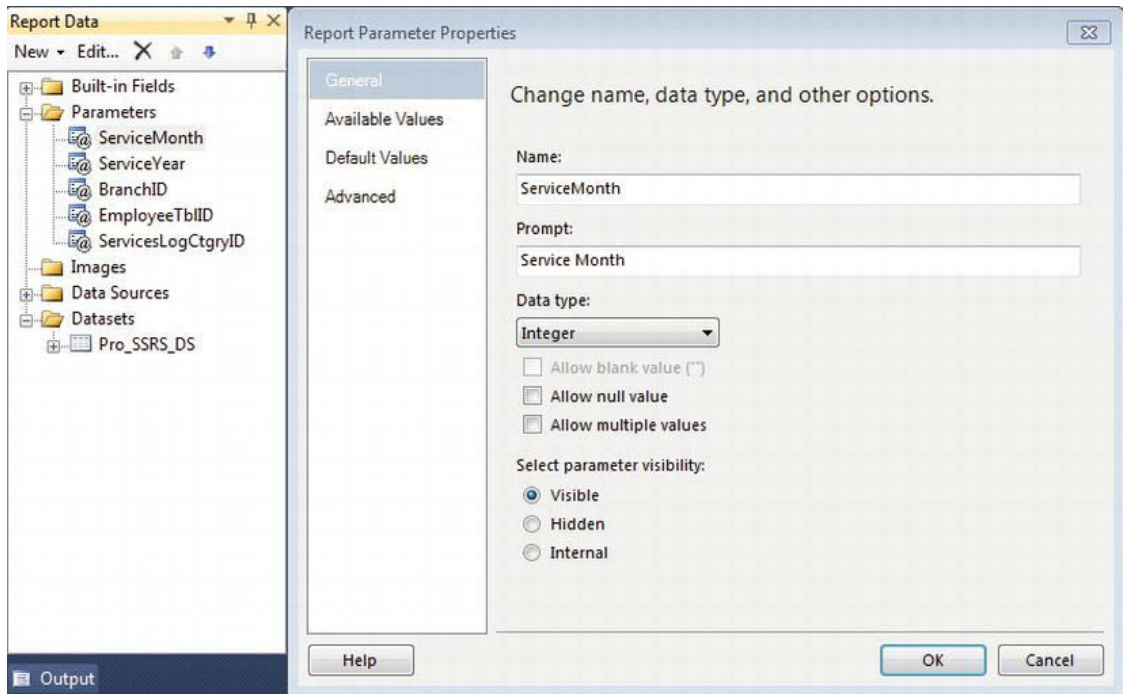


Figure 3-11. Report Parameters dialog box

Figure 3-11 also shows the `Allow Multiple Values` checkbox, a new SSRS feature for SQL Server 2005. Multivalue parameters allow users to select all values or combinations of values to be used within the report to limit the data that is displayed. When multiple values are chosen, they are passed to the query or stored procedure as a string array. It is important to note that multivalue parameters require the following special considerations when implementing in your reports:

Does not accept NULL values: This is important when deciding which parameters to make multivalued, as it will influence the design of the underlying query or stored procedure. In this case, you built logic into the Emp_Svc_Cost stored procedure to accept NULL values and return all data when NULL was passed in from a parameter. You will have to modify this stored procedure to work with multivalue parameters.

Will be evaluated as a String: Since a multivalue parameter returns a comma-separated string, you will need to consider the data type assignment for the stored procedure parameters—the report parameter and the query or stored procedure parameter need to have the same data type to work properly.

Affects performance: Multivalue parameters are best used when there is a relatively small list of values. Choosing to allow users to select a range of years—for example, “2010,2011,2012”—is much better than allowing them to select 1,000 patients based on their IDs, as these will all be passed into the stored procedure as a comma-separated string value to be evaluated with the IN or EXISTS clause.

Cannot be used in filters: Unlike single-value or nullable parameters in SSRS, multivalue parameters can be used only to pass back to the query or stored procedure, so you can’t use them to limit with report filters.

Requires string manipulation logic in stored procedures: Stored procedures do not evaluate multivalue parameters correctly, so using IN (@MyReportParameterArray) in a stored procedure, for example, will not return the expected result. This has been an issue with SQL for a long time, and numerous ways, both good and bad, exist to work with multivalue string arrays in stored procedures. Two possible choices are user-defined functions (UDFs) and dynamic SQL. In Chapter 6, which covers the building of a deployable report, we will discuss how to use a special UDF that parses the multivalue report parameter into a table that will work effectively to limit the result set to exactly what is expected.

Setting up Filters

Like parameters, report filters can limit the results of data on a report; however, you don’t necessarily have to use them in conjunction with a parameter. In fact, filters, which can be defined at many points in the report, evaluate an expression and filter the results based on that execution. Filters take this form:

<Filter Expression> <Operator><Filter Value>

An example of a filter is one that limits the data on a report to a specific user or that is based on user input from a parameter value.

Chapter 11 demonstrates how to use a filter that limits the report based on a built-in Global collection, which includes the username of the person executing the report. Filters are beneficial because once the report is rendered you can use them in conjunction with parameters to limit the data in the report without re-querying the data source. In Figure 3-12, you can see a filter that limits the data displayed based on a parameter called User. The logic is this: if the parameter value for User is equal to a field value of User, then include only those records where they match. Otherwise, include all records. Parameters and filters are included as elements of an RDL report file.

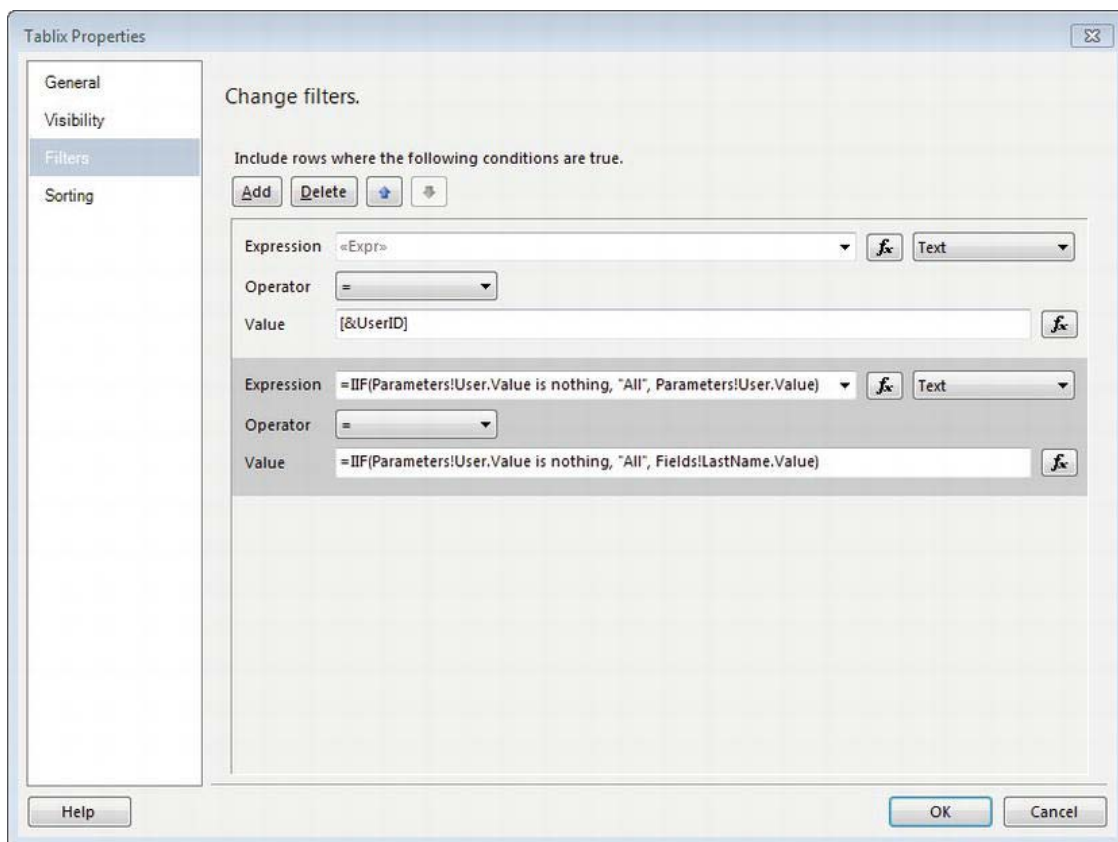


Figure 3-12. Sample filter on a table data region

Listing 3-3 shows the sample RDL elements of a filter.

Listing 3-3. Parameter and Filter RDL Elements

```
<Filters>
  <Filter>
    <FilterExpression>=IIF(Parameters!User.Value is nothing, "All", Parameters!User.Value)
    </FilterExpression>
    <Operator>Equal</Operator>
    <FilterValues>
      <FilterValue>=IIF(Parameters!User.Value is nothing, "All",
Fields!LastName.Value)
      </FilterValue>
    </FilterValues>
  </Filter>
</Filters>
```

Expressions

Throughout this section, you'll use fields from the dataset to create sample report segments. Because the values from the fields are derived from *expressions* that are essentially VB.NET code, we will cover them now because they play a crucial role in the report design process.

You can use expressions to produce a value for any report item that uses them. In SSRS, you can assign expressions to almost any report property, from formatting such as color or padding to the value of a textbox. A simple expression such as that of a field assignment is commonly used while designing reports. In fact, every time you add a field to an area of a report, it's automatically converted to an expression, like so:

```
=Fields!FieldName.Value
```

An expression is signified by prefacing its content with the equal sign (=). You can also concatenate expressions with other functions and literals. We will show several examples of expressions throughout the book. We will list several sample expressions here and show you how to assign them to report items:

`=Parameters!ParameterName.Value`: Used to assign the value of a parameter to a report item such as a textbox or cell in a table.

`=IIF(Fields!FieldName.Value > 10, "Red", "Black")`: Used for conditional expressions. In this case, it would set the text color for a property to red if the value of `FieldName` were greater than 10, black otherwise.

`=Fields!FieldName1.Value & " " & Fields!FieldName2.Value`: Used to concatenate the value of two fields.

`=Avg(Fields!FieldName.Value)`: Used to aggregate functions such as Sum, Avg, Min, Count, and Max that return the aggregate, minimum, or maximum value of the specified field.

`=RowNumber(Nothing)`: Used to maintain a running total for the row numbers in a report. "Nothing" in this case is a scope parameter passed to the function indicating a grouping or dataset. The scope parameter could be a group name or dataset, in which case a new row count would begin at the end of each group or dataset.

In SSRS for SQL Server 2005, the expression builder application used inside the report development environment was rebuilt to give it the type of functionality needed to assist users create useful expressions easily. Thankfully, all releases since then have retained the more advanced expression builder, which lists most of the common functions, along with examples using their syntax; in addition, they are categorized by type (Text, Conversion, and Date & Time). This makes it much quicker to find the right function and place it as part of the expression you are building. Another great feature that developers have become accustomed to, and frankly should not have to live without, is IntelliSense, a contextual, in-line command completion feature. As you can see in Figure 3-13, as you type an expression (in this case a field value expression from your stored procedure), you are prompted with all the possible selections based on that expression. Once the expression is complete and syntactically correct, you can click OK to make the expression part of the report object where you have associated it. If any syntax errors exist, a standard red underline indicates the problem, and a mouse-hover over it will display the type of error, in most cases "Invalid Syntax." This is a quick introduction to expressions in SSRS, but in Chapter 6, we will use them to perform tasks such as setting default parameter and property values at runtime.

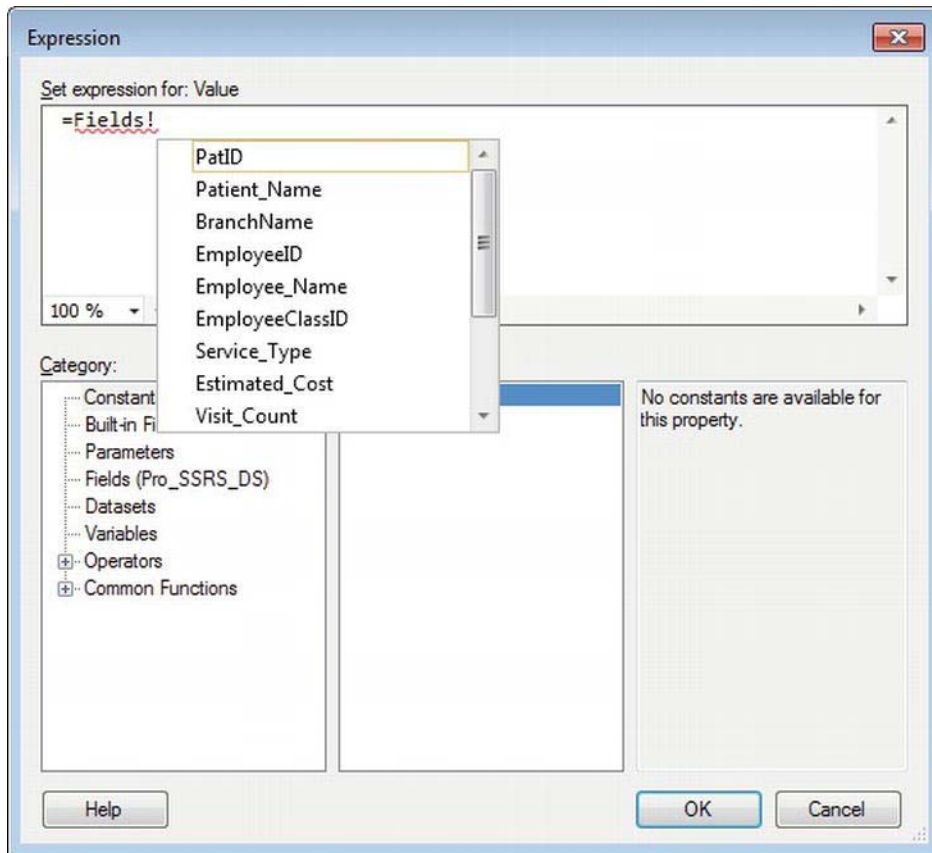


Figure 3-13. IntelliSense: Assigning an expression

Summary

At this point, we have only scratched the surface of what the development environment has to offer. In this chapter, you were introduced to the development environment and some of the basic elements of Reporting Services design. You learned that each report consists of defined elements based on a defined schema in the RDL, which gives SSRS the advantage of standardization. We covered some of the report objects that make up reports such as Data Sources and Datasets. In Chapters 4 and 5, you will learn about all of the objects that will be used to display or represent data returned by our datasets such as the Textbox, Rectangles, and Data Regions like Tablix, List, and Matrix.

Laying Out a Report

Now it is time to delve into the area in the IDE in which you'll probably spend the most time: the Design tab. The real creative magic begins here, and we don't mean because a wizard or two might be involved.

Different report requests and target audiences mean that the look and feel of each report will not be uniform. One user might expect drill-down functionality, while others might need full, detailed listings of data for printing. Whatever the case, SSRS provides many tools in the Toolbox for rapid and efficient building of high-quality reports that can be immediately deployed from within BIDS. In the following sections, you'll take the sample data and put it to use as you explore the functionality of each of the available tools and data regions.

For each object we demonstrate, we'll give the graphical representation of the design environment as well as its RDL counterpart. Note that defined sections of the RDL file contain every aspect of a report, from the general layout to pagination. This is important because it's often easier to work directly within the RDL file to alter a report. As we show how to add functionality to the sample report projects, we'll point out sections of the RDL files where the graphical report design is converted to code.

Setting Up Pagination

To begin, look at the general report properties for the new report. While on the Design tab, select Report and then Report Properties from the drop-down menu on the toolbar.

There are four selections in the Report Properties dialog box: Page Setup, Code, References, and Variables. For now, you're concerned with the Page Setup selection. As you can see in Figure 4-1, the Page Setup dialog contains property settings for pagination, such as page width and margins. It's only since SSRS for 2008 that we've been able to select standard page formatting options such as Portrait or Landscape orientation. Your selection sets the Width and Height property values automatically. Because a number of reports will be printed in landscape format, choose Landscape. Leave all margins at 1 inch each. If you are still on SSRS 2005, you will need to set the height and width properties to represent Landscape characteristics like 11 inches in width by 8.5 inches in height.

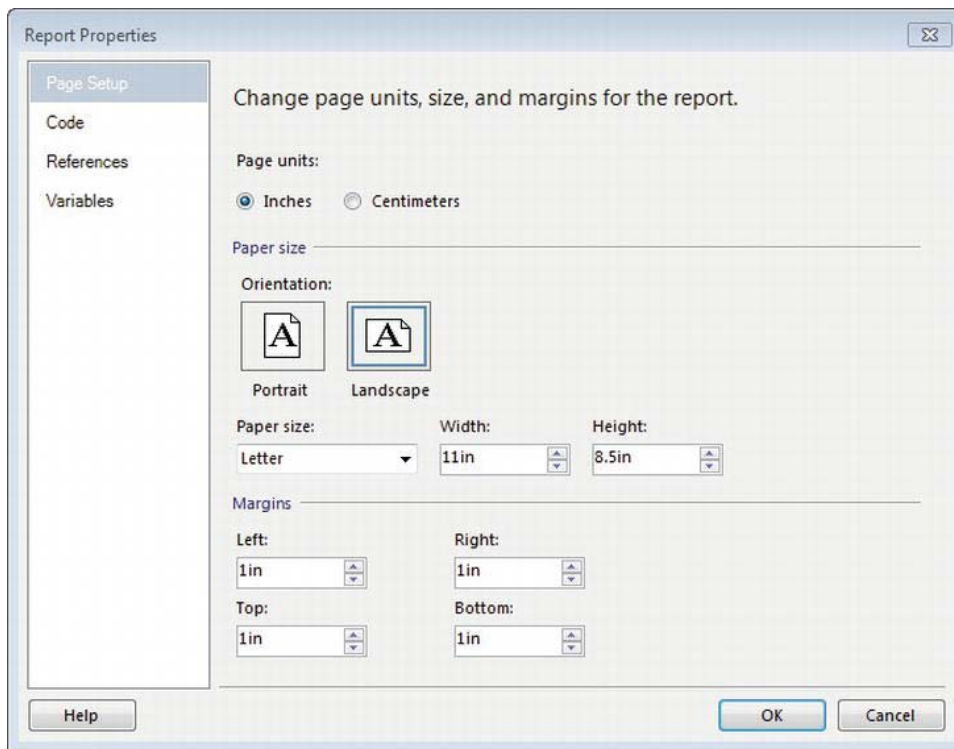


Figure 4-1. Report Properties Layout tab

If you are using 2005, the Layout tab will also have a Columns setting, where you can specify the number of columns. This section was removed in 2008, but it is still a property of the Report. We use multi-column reports frequently in the medical industry, primarily for printing labels.

Using Report Objects

In the next few sections, you will look at the report objects to see their basic functionality and learn how you can tap into each one's unique versatility when developing reports. At this point, if you have not already done so, open the Pro_SSRS solution that we have provided with starting points and completed samples, and refer to it as you work through the individual object sections. We will cover the following report objects:

List: This is a free-form container object for a single data grouping.

Textbox: The textbox report in SSRS 2008 is probably the most long-awaited enhancement. In previous version of SSRS, textboxes were nothing more than containers for strings of uniform literal text and/or field values from data sets; all had to have the same formatting. Since the SSRS 2008 release, this is no longer the case.

Table: This is used for tabular reports with rows and columns, but provides single or multiple data groups.

Rectangle: Like the List, the Rectangle is a container, although it provides no data groupings.

Matrix: This report object, like the Table, provides multiple grouping levels, but lays data out as a cross-tab or pivot-style report.

In SSRS for 2008, Microsoft introduced the Tablix control, which at its core is a conglomeration of row-level grouping and column-level grouping. Before SSRS 2008, it was difficult to create a report that combined these two levels of grouping. You can use the Tablix properties in three controls, List, Matrix, and Table, creating on-the-fly column or row groups at any level within the three report objects. RDL code for Tablix support has been added as well, as you will see as we go through the report objects that support Tablix properties. In Chapter 6, you will see more detail on adding Tablix properties to a report.

Implementing a List

The List data region is one of the two free-form container objects that allow grouping of data, the other being Rectangle. The two objects are similar in that they can contain other report objects and data regions. Free-form data regions don't constrain the layout of fields to a fixed format; the person creating the report is responsible for aligning the objects.

Because the List contains a grouping level, you can use the List data region only with a single dataset. Note that the List data region displays one record at a time from the dataset based on this grouping. By default, no grouping is assigned to a list.

To learn how you can use the List data region with the `Emp_Svc_Cost` stored procedure, which returns detail records for the number of visits for patients, you will add a List to the design area and drag fields from the dataset into it. You will use `Employee_Name`, `Patient_Name`, `Visit_Count`, and `Estimated_Cost` to show total visits and cost for each patient/employee combination.

In the `Pro_SSRS` solution, the list sample's starting-point report is called `List Start`. This report has a data source and dataset, called `emp_svc_cost`, already defined for the localhost server, which should match your environment if you are using BIDS to connect to your local SQL Server.

To begin, open the `List Start` report, and click the Layout tab. Before we get started on creating the report, click on `Report > View > Ruler` to make the ruler visible on the designer. Next, add the List report object to the report by double-clicking the List object in the Toolbox. The List control is automatically added to the upper-left area of the design grid. Grab the lower-right corner of the List and drag it down until it is approximately 7 inches wide by 1 inch in height.

Next, select the Datasets in the Report Data toolbar, expand the `emp_svc_cost` dataset, drag the fields in the following list to the design area, and place them in the List data region:

- `Patient_Name`
- `Employee_Name`
- `Service_Type`
- `Estimated_Cost`
- `Visit_Count`

Now add the Sum function to the Visit_Count and Estimate_Cost fields so that each field will be of the syntax =Sum(Fields!fieldname.Value). If you had added a grouping level to the list (you will do that shortly), the Sum function would have been automatically added to these fields.

Next, size the fields so that when you're finished, the report looks like Figure 4-2. In Visual Studio 2010 and BIDS, as you work with the layout of report objects, the design environment has been enhanced to control automatic alignment as you drag and drop objects, so it is far easier to design professional-looking reports. At this point, however, you aren't concerned with beauty, but with functionality.

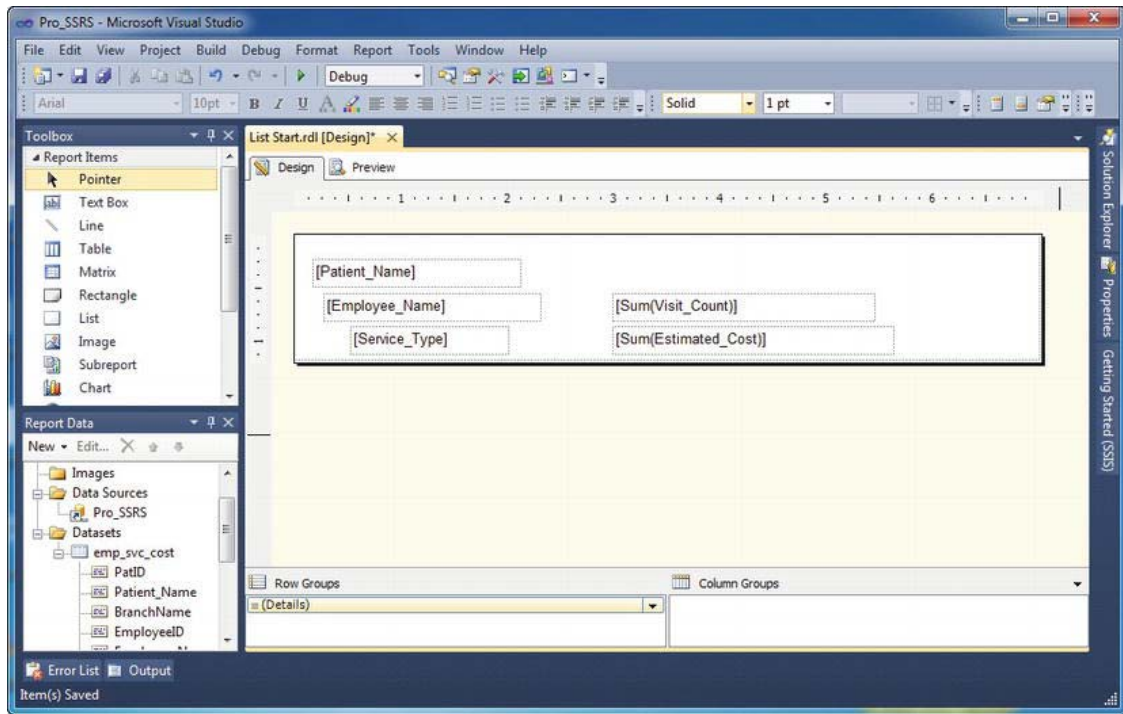


Figure 4-2. List data region with ungrouped fields

When you click the preview button, the report will generate and display visit information for each patient and employee (see Figure 4-3). Notice that the sum of the fields Visit_Count and Estimated_Cost is the same for each record. Each sum amount is the same for all patients and employees because you have not yet defined any grouping for the List itself. You will do that next.

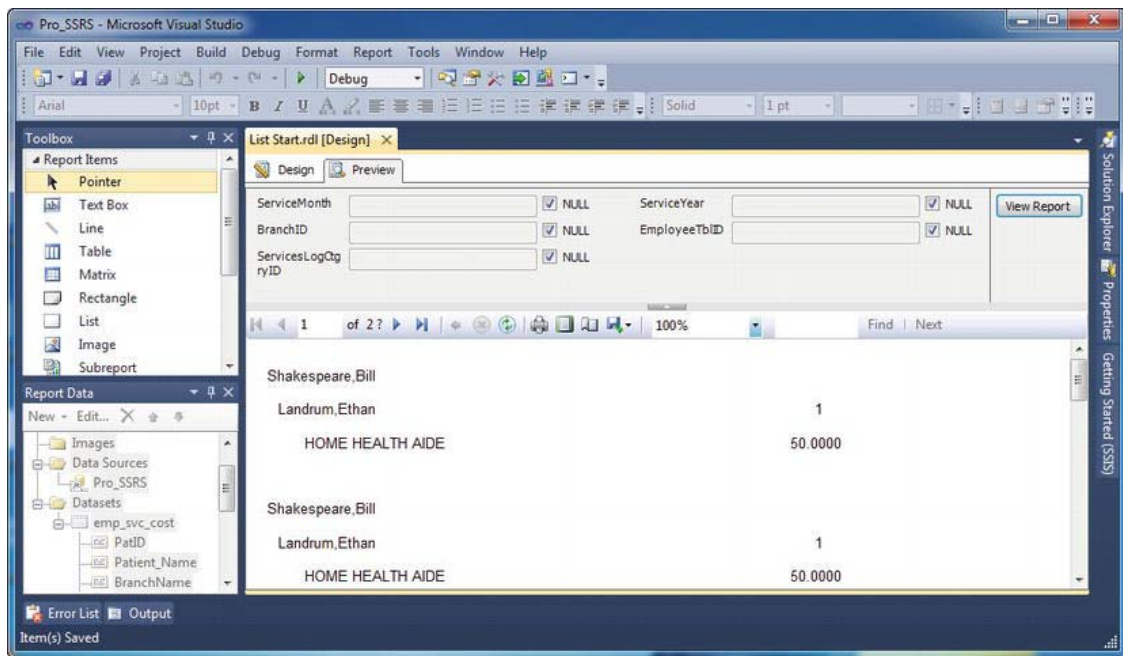


Figure 4-3. Preview of the List data region

For the List we added, there is a default (Details) row grouping as evidence in the Row Groups area of the report. However, no field value is currently assigned as the group; it is simply blank. To add grouping, click the down arrow on the Row Groups (Details) and select Group Properties. Add two fields by clicking Add in the Group Expressions area—Patient_Name and then Employee_Name—as group expressions.

The preview of the new List with grouping levels now shows the correct number of visits for each patient and employee combination. However, notice that in Figure 4-4 Bill Shakespeare is showing up multiple times, with 37 visits for employee Bailey McDonald and eight visits for employee Sherri McDonald, for example. What you really want to see is Bill Shakespeare once with all the employee and visit information grouped in the figure. You could remove the grouping for the employee from the List, but that would cause the report to group all the patients' visits under a single employee and reflect the data inaccurately. In former versions of SSRS, only a nested List would achieve this level of multiple groupings and the report developer may have needed to look at using an alternative report object, like a table—that would have got the level of desired grouping, but would have lost the free-form feel of the report. Since SSRS 2008, however, the List object includes Tablix properties. In other words, when you add a List object from the Toolbox onto the design surface, a Tablix is created. The Table and Matrix objects also share the Tablix Properties are implemented as a Tablix. With Tablix properties, it is possible to create a parent group above the default Details Group and add the patient name information there, so the desired fields, Patient_Name and Employee_Name, can now be grouped hierarchically.

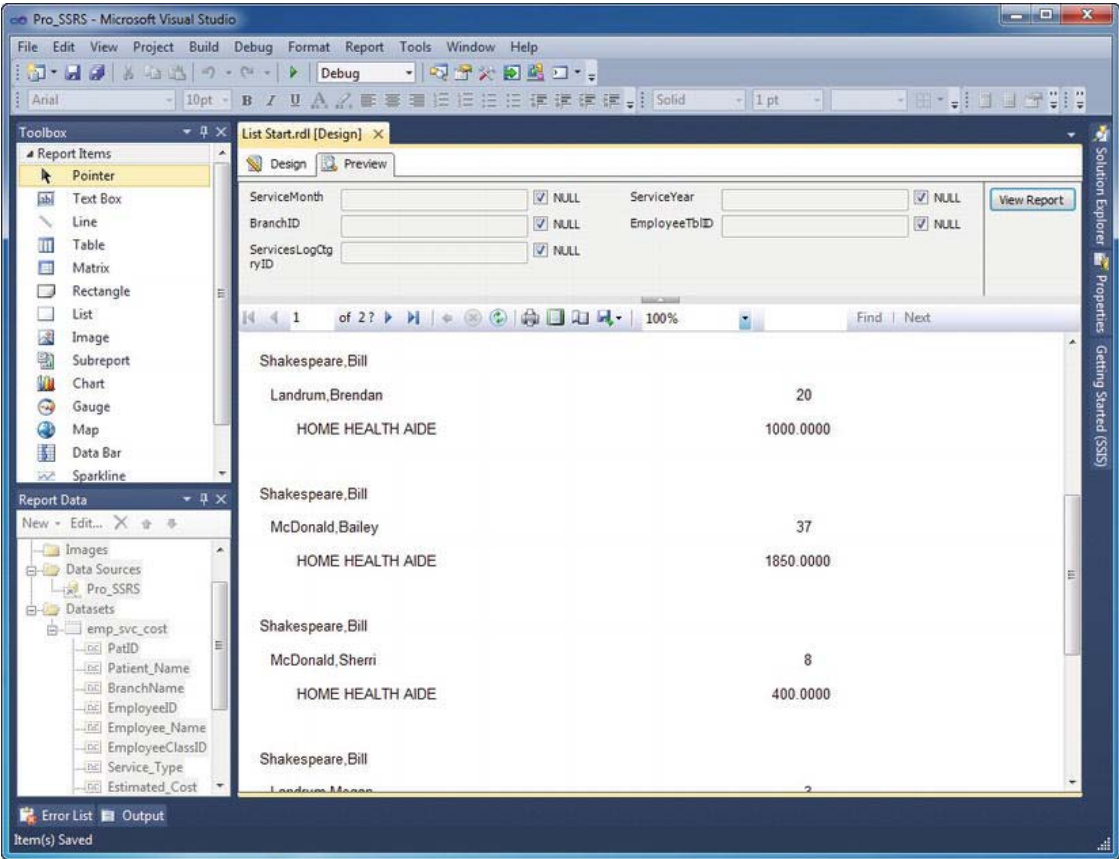


Figure 4-4. Previewed report with duplicate patient

In SSRS 2005, you would have had to add two Lists, one nested in the other, to produce output that would allow for multiple groupings in one container object. We showed that approach in the previous edition of this book. However, with the introduction of Tablix properties, we can simply add another grouping above Employee_Name. To do this, right-click the details row of the List box where we added the fields, and select Add Group; then Parent Group under the Row Group submenu, as shown in Figure 4-5. For the group, select Patient_Name as the expression. You will notice that a new column is added for your new Patient_Name group. Select the Patient_Name textbox that we added earlier and delete it from the details section.

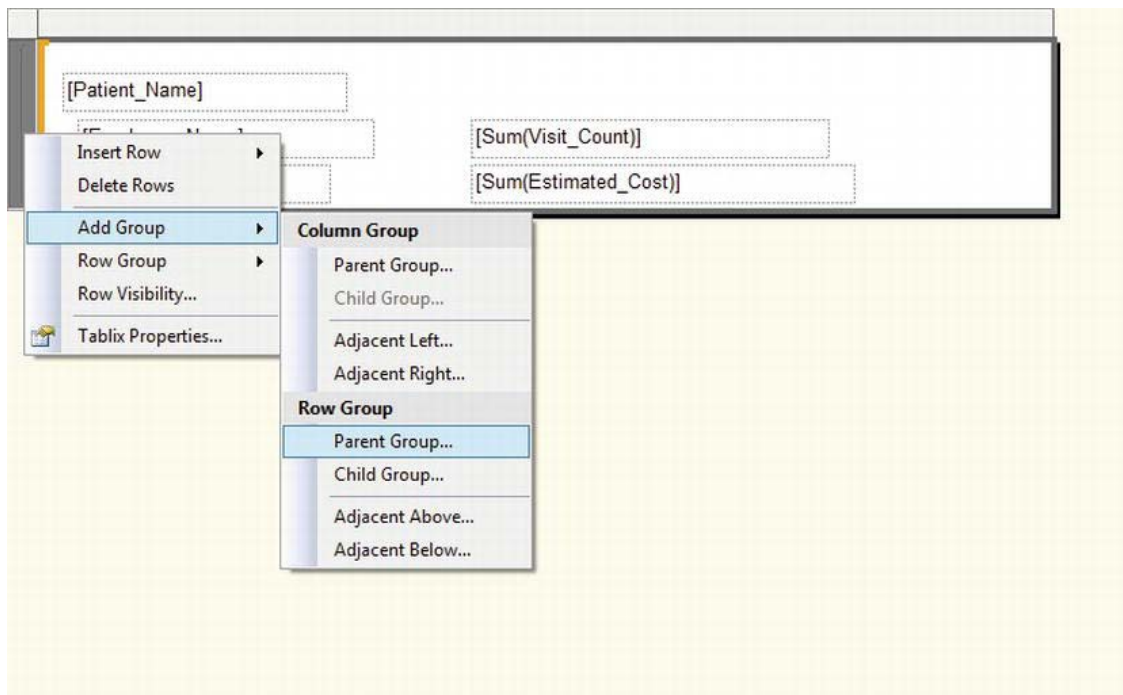


Figure 4-5. Adding the *Patient_Name* grouping to the List

Now, after a few modifications of spacing and alignment when you preview the report grouped by *Patient_Name* and the child group *Employee_Name*, you can see that multiple employees are associated with a single patient. Along with each employee displaying the correct total number of visits and estimated cost (see Figure 4-6).

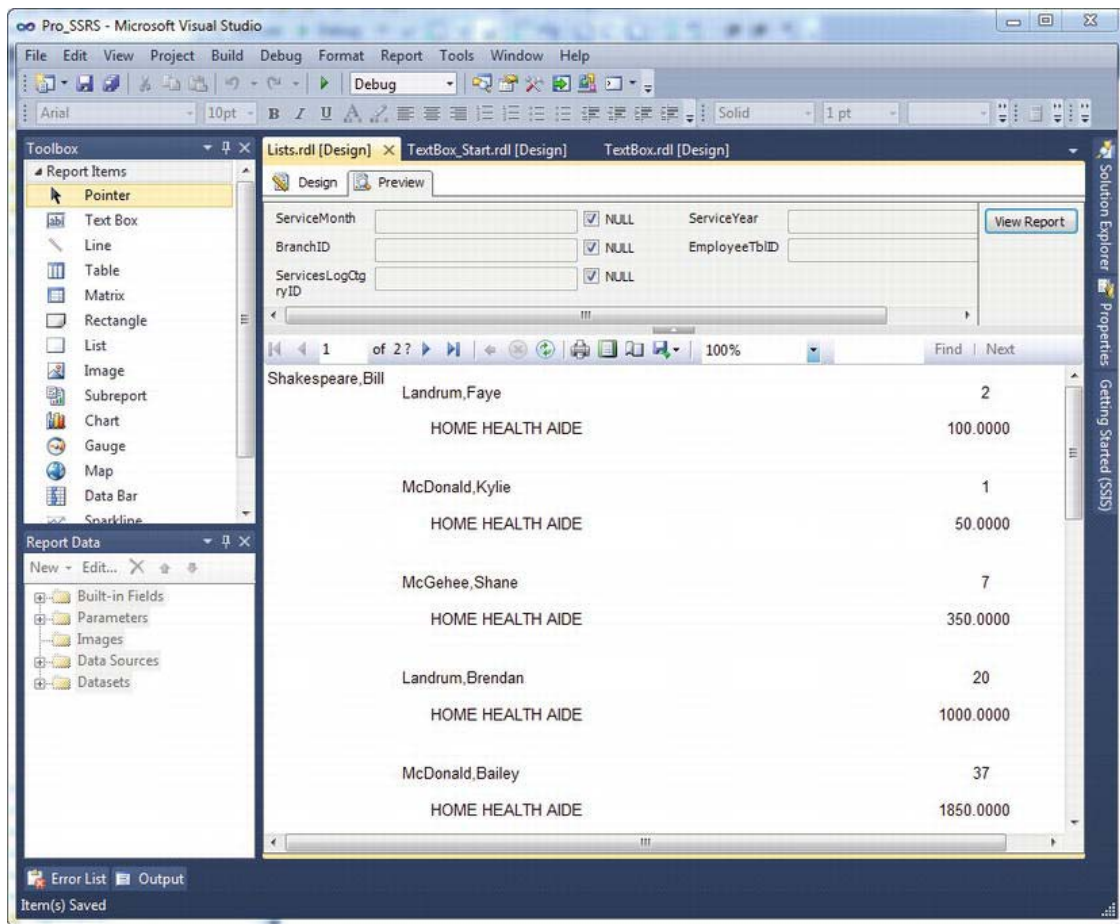


Figure 4-6. Previewed report with parent group

Listing 4-1 shows a section of the RDL file with a sampling of the List data region you just created. Notice in the XML schema that the `<List>` element encapsulates everything that has been graphically added to the List data region, including all formatting and grouping. As you can see in the RDL, the List control we selected inherits properties of the new Tablix functionality.

Listing 4-1. RDL List Section

```
<TablixRowHierarchy>
  <TablixMembers>
    <TablixMember>
      <Group Name="Patient_Name">
        <GroupExpressions>
          <GroupExpression>=Fields!Patient_Name.Value</GroupExpression>
```

```
</GroupExpressions>

</Group>
<TablixHeader>
<Size>1.08333in</Size>
<CellContents>
    <Textbox Name="Patient_Name1">
        <CanGrow>true</CanGrow>
        <KeepTogether>true</KeepTogether>
        <Paragraphs>
            <Paragraph>
                <TextRuns>
                    <TextRun>
                        <Value>=Fields!Patient_Name.Value</Value>
                        <Style />
                    </TextRun>
                </TextRuns>
            <Style />
        </Paragraph>
    </Paragraphs>
    <rd:DefaultName>Patient_Name1</rd:DefaultName>
    <Style>
        <Border>
            <Style>None</Style>
        </Border>
        <PaddingLeft>2pt</PaddingLeft>
        <PaddingRight>2pt</PaddingRight>
        <PaddingTop>2pt</PaddingTop>
        <PaddingBottom>2pt</PaddingBottom>
```

■ **Note** To access the full RDL file from within BIDS, select View with the Design tab open, and then Code from the drop-down menu. If you are working in Visual Studio 2010, press F7 and use Shift+F7 to toggle between Code and Designer modes.

The completed report for the List object is called `Lists.rdl` in the Pro SSRS project.

Implementing a Textbox

The Textbox control had a major upgrade from SSRS 2005 to 2008, but nothing has changed since. Thankfully, it now supports rich formatting for text, not to be confused with Rich Text Formatting. What is possible now was not possible, or at least very difficult, to do in SSRS 2005. You can now format text blocks individually inside a single textbox as well as combine fields from datasets using placeholders within the textbox. This is all accomplished via HTML tags that can be imported as expression values for the textbox itself or contained within a dataset and rendered with the report.

To learn how to create a textbox that has rich formatting, start with the `TextBox_Start.rdl` report that contains an empty Textbox control. Open `TextBox_Start.rdl` in the `Pro_SSRS` solution. The initial report will look like Figure 4-7.

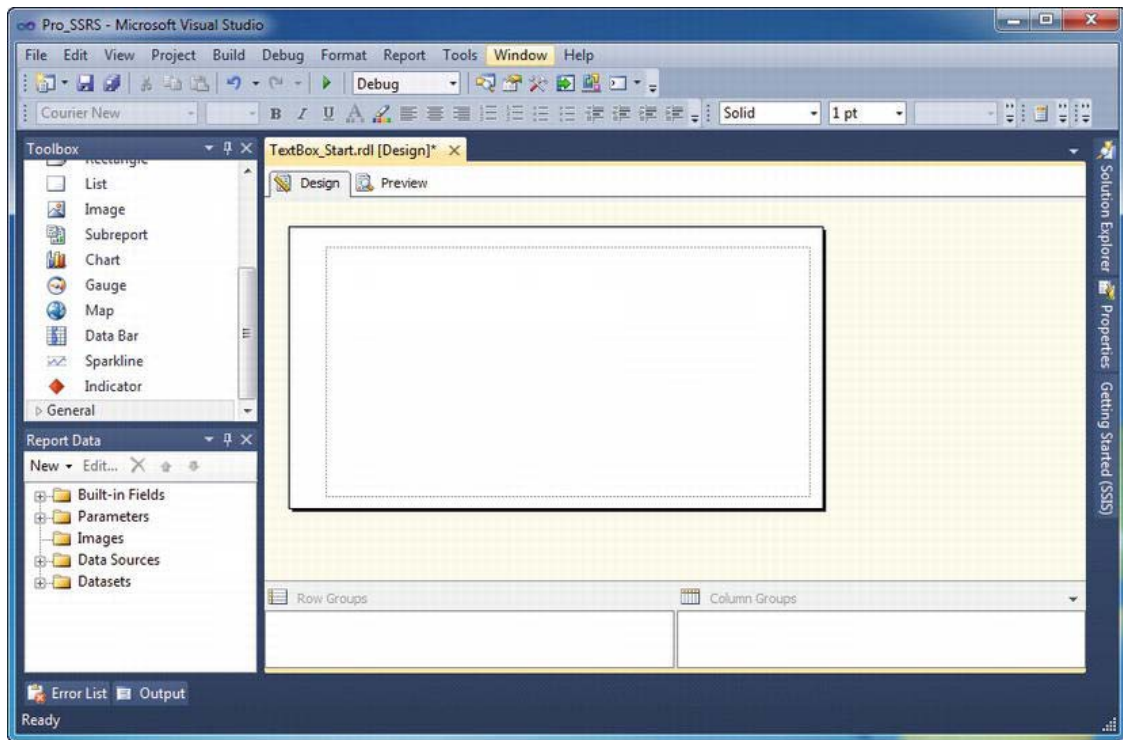


Figure 4-7. *TextBox_Start* report with blank textbox

For this example, let's assume that, as a report designer, you have been tasked with merging literal text with data in a form letter that will be sent to patients as a survey. The survey, which we will complete in Chapter 6, will include information about a patient's specific visits, but must otherwise be a generic template with the same questions for each patient. The report will also include bold, underline, and italic font properties in a single Textbox.

The first thing to understand about textboxes since SSRS 2008 is *placeholders*. A placeholder is simply a location in the Textbox that will contain individual portions of the Textbox and can have custom formatting for the value or expressions inside it. Each placeholder has its own properties as well, such as Tooltips, Actions, and Alignment.

First, click in the textbox of the `TextBox_Start` report to select it. Then, right-click and select `Create Placeholder`. You will see the Textbox properties, as in Figure 4-8.

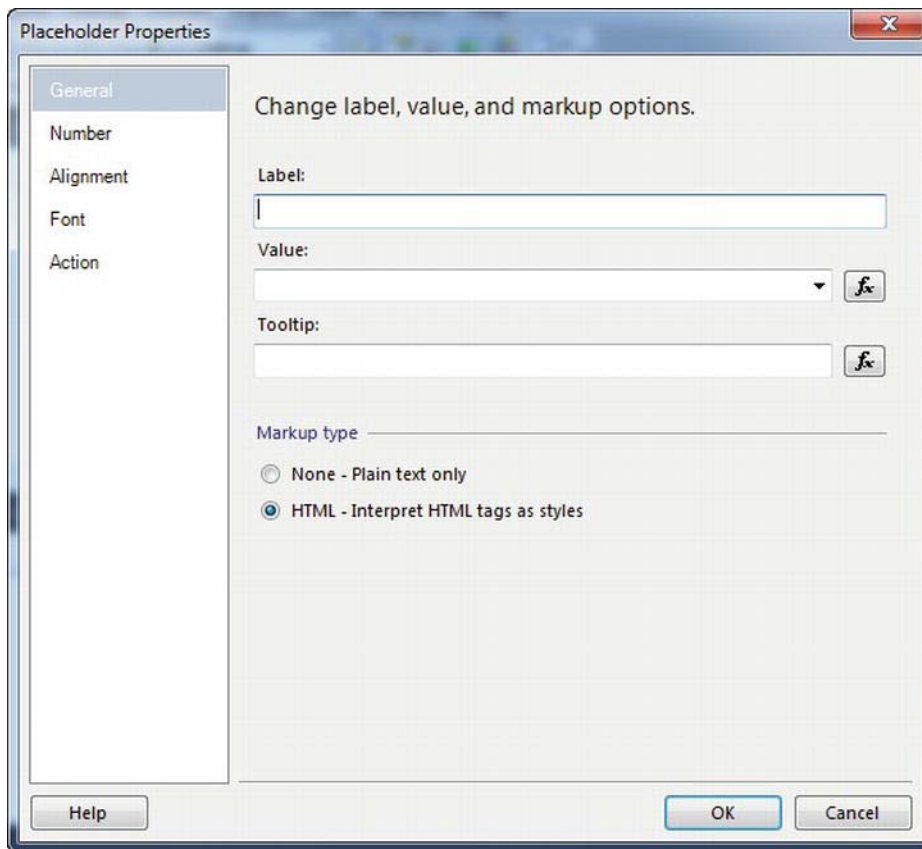


Figure 4-8. Textbox Placeholder properties

For this demonstration, you will add a simple date placeholder, a greeting, and a data field, each with different formatting styles. In chapter 6, you will expand on this by creating a patient survey form letter that combines multiple objects like Rectangle and Table to deliver a detailed report of services rendered along with the survey. For a simple Textbox like this one, no grouping is available for the dataset, Pro_SSRS_DS, so you will use the First function to extract the patient name from the dataset for this example.

With the Placeholder Properties dialogue box still open, type “Today” in the Label field and “=NOW()” in the Value field. One important note on formatting is that if HTML is selected on the General tab, other date-formatting rules on the Number tab will not be applied. Therefore, for this placeholder, select None – Plain Text Only as in Figure 4-9.

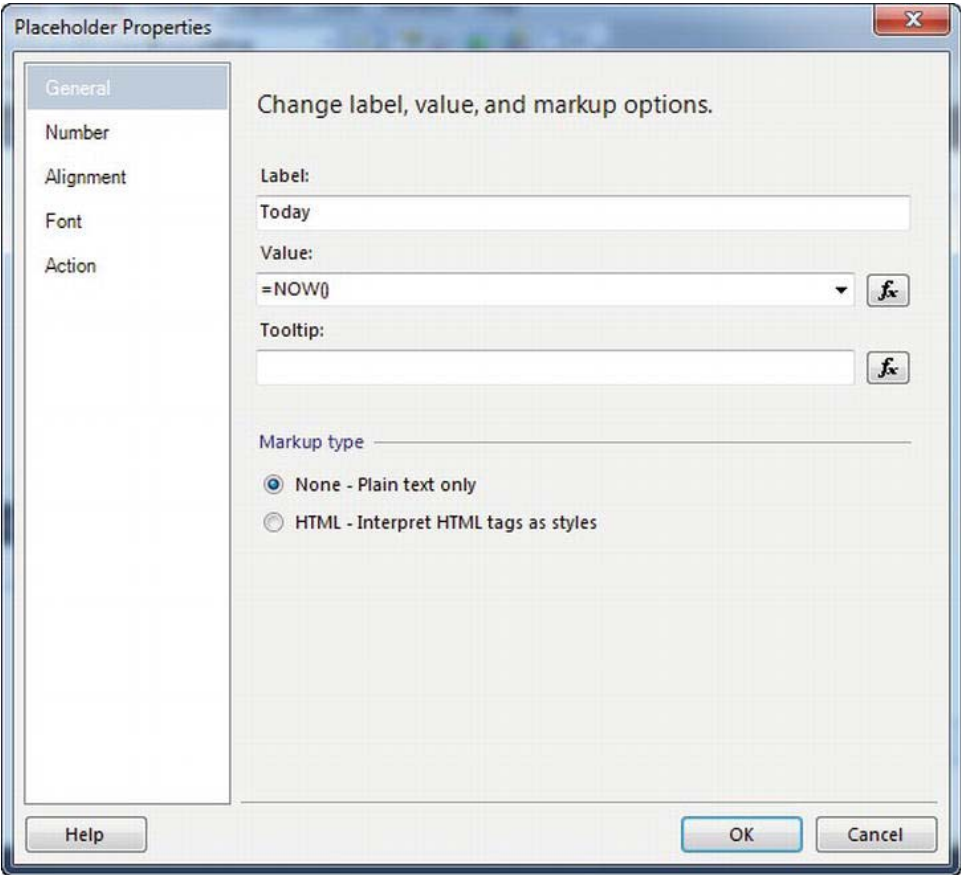


Figure 4-9. Value properties for the first textbox placeholder

On the Number tab, select Date for the Category and for the Style select Monday, January 31, 2000 as in Figure 4-10, and then click OK.

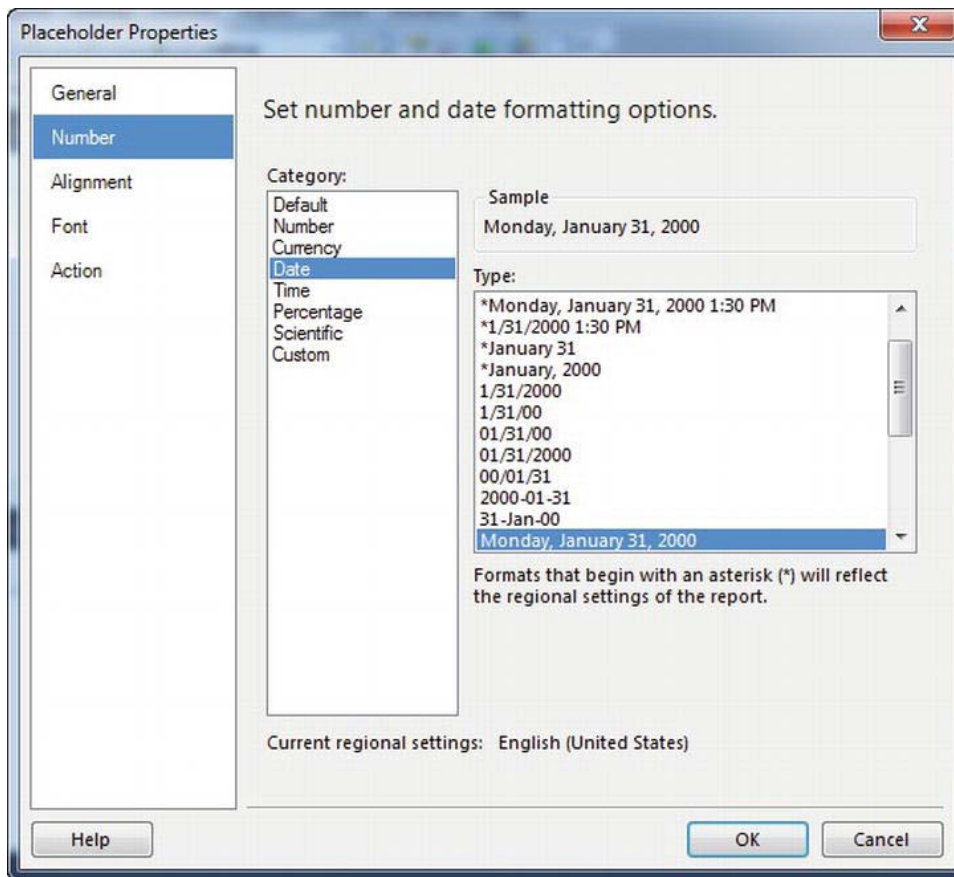


Figure 4-10. Plain text date formatting for the textbox placeholder

Back on the Design tab, click below the placeholder we just added to go to the next line in the Textbox. Type the word “Dear” directly into the textbox and follow it with a space. This will serve as the greeting for the letter. Now, right-click the location directly after the space you typed, and select Create Placeholder again. For the label, type “Patient Name.” For the expression, open the expression builder by clicking the expression button to the right of the value field. Click DataSets in the Category box and in the Item box select Pro_SSRS_DS. In the Values box select First(Patient_Name). Notice, as in Figure 4-11, that the value is expanded to be =First(Fields!Patient_Name.Value, "Pro_SSRS_DS"). You will notice here, and throughout the book, that in SSRS 2012, full expression values for fields have been condensed for ease of use and clarity. Fields!Patient_Name.Value, for example, will often be displayed as simply [Patient_Name] in design mode.

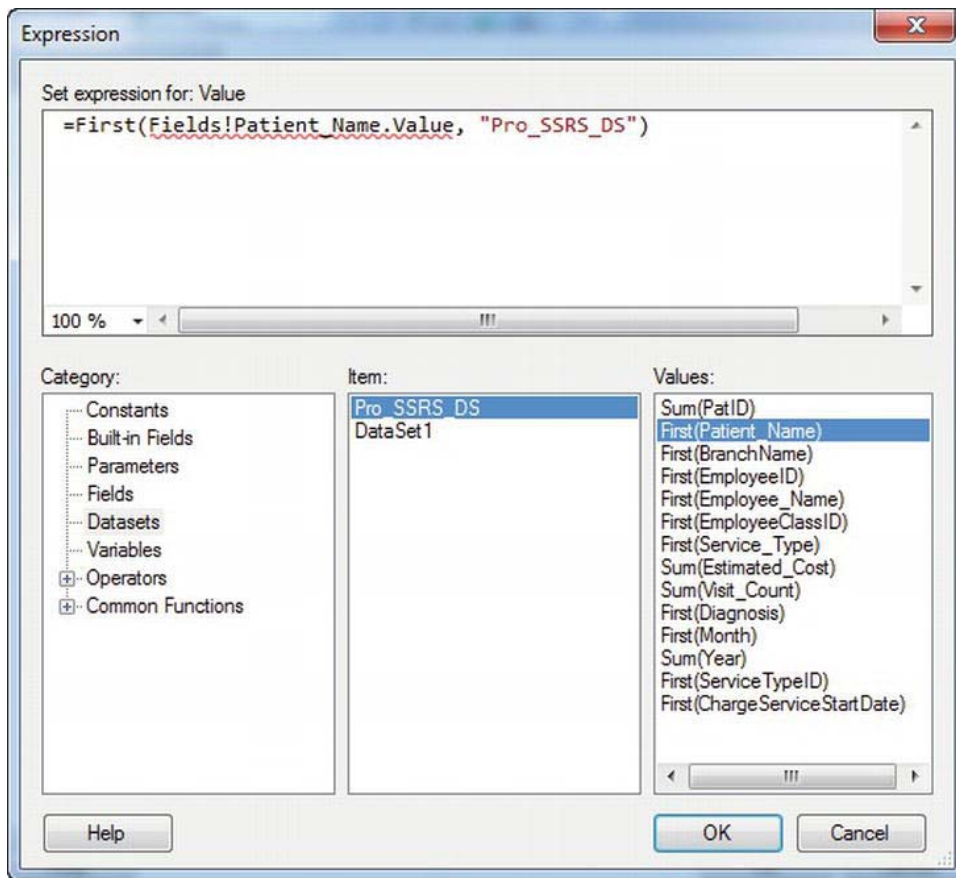


Figure 4-11. Patient Name value for the textbox

Rendering the report at this point does not effectively show how you can have multiple formats in a single textbox. However, adding individual formatting at this point is as easy as selecting your placeholders and choosing whatever format you desire. Highlight the Today placeholder, making sure that the entire label is selected. Next, click the italic button from the toolbar. Then, select the Patient Name placeholder and click the bold button. Figure 4-12 shows a preview of a single Textbox object with multiple formats.

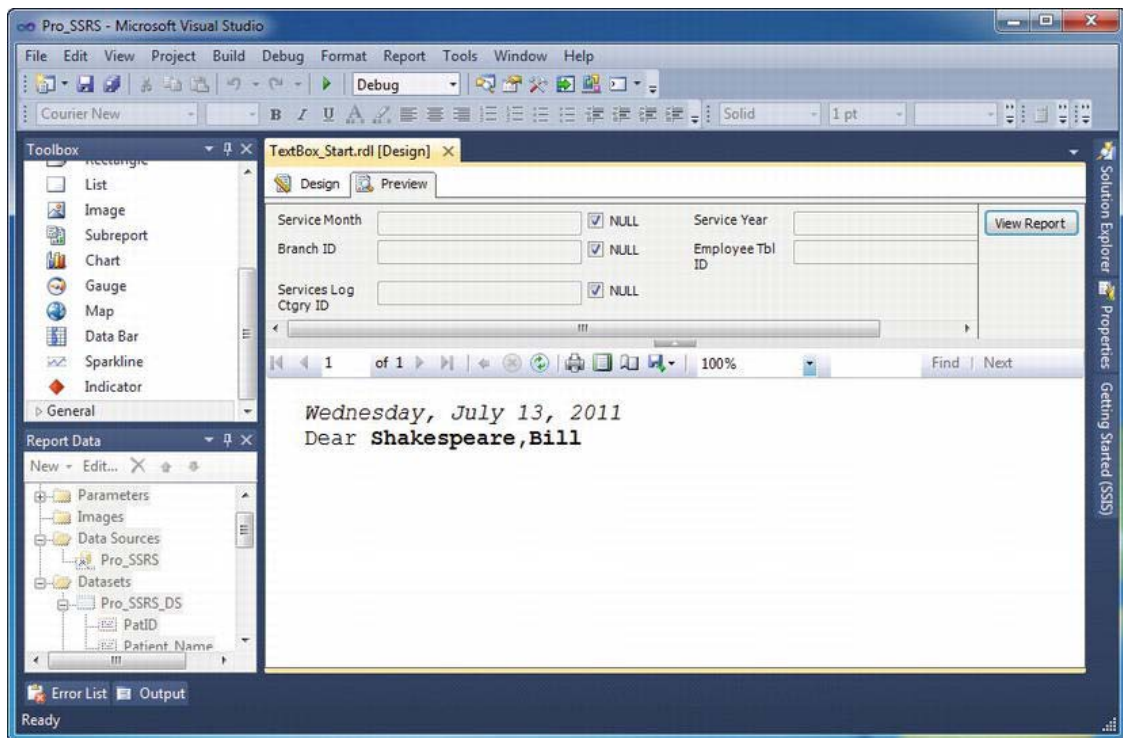


Figure 4-12. Output of multiple formats for a single textbox

As stated earlier, even custom formatting as simple as this was not available in SSRS 2005, so SSRS 2008 represents a huge leap forward in report design. In the next chapter, you will produce reports that are more sophisticated by importing HTML formatting directly from a SQL table, as well as records from multiple datasets, to create a full custom patient survey letter.

The completed report for the Textbox object is called `TextBox.rdl` and is in the `Pro_SSRS` project.

Implementing a Table

The Table data region provides a means of organizing data into tabular rows and columns with possible multiple grouping levels. Every Table data region has, by default, a row that contains detail records as well as Table headers and footers. You can group tables on individual fields from a single dataset or with expressions that might combine multiple fields. Tables make it easy to design a uniform report because of the structured nature of the Table. Fields from the dataset are simply added to a cell within the Table, and when the report is rendered, it's automatically formatted. The List data region provides much of the same functionality as tables, but you have to position and align fields manually.

The starting-point report for the Table report object that we will demonstrate in this section is called `Table_Start.rdl`. You will find it in the `Pro_SSRS` project, in the Source Code/Download area for the book on the Apress site. Open the `Table_Start.rdl` file in BIDS to begin.

As you did with the List, you will begin on the Design tab and double-click the Table in the Toolbox. This adds a Table to the report design area. Next, drag the same fields you used for the list example to the detail row of the table—Employee_Name, Service_Type, Estimated_Cost, and Visit_Count as detail information and Patient_Name for the group. To add the first four fields to the detail row, you need to add a fourth column to the Table. You accomplish this by right-clicking the bar at the top of the middle column and selecting Insert Column to the Right. You now have four equally sized columns to which you can drag the four fields so that the report appears as it does in Figure 4-13, with the fields placed in the detail row of the Table. You will wait to add the Patient_Name field until you add a grouping level. Notice that the column headings for each of the fields are added automatically when the fields are dropped into the individual cells.

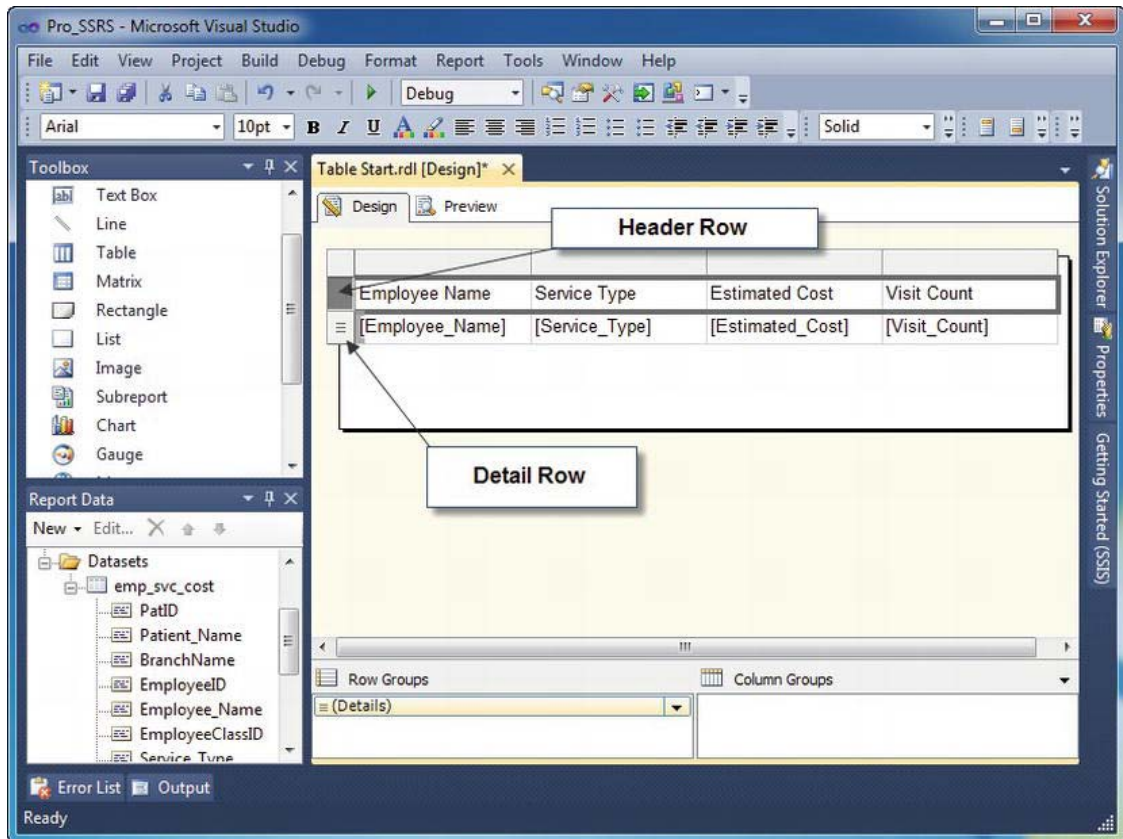


Figure 4-13. Table data region with detail rows

To add the Patient_Name field to the Table so that the report has the same functionality as the list you created previously, you need to insert a group in the Table. To do this, simply drag the Patient_Name field in the emp_svc_cost dataset down into the Row Groups section and release it just above the (Details) row group as shown in Figure 4-14 below.

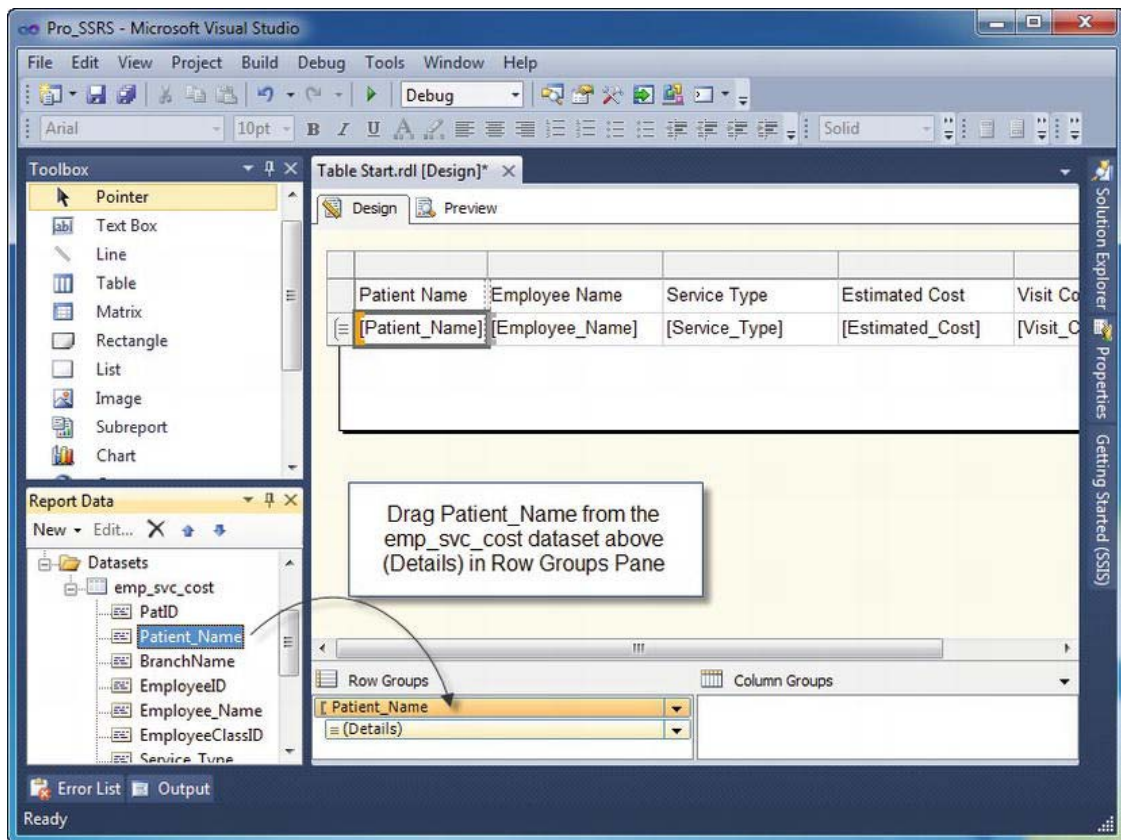


Figure 4-14. Creating a Row Group using drag-and-drop

You can view the Group Properties by clicking the down arrow to the right of the Patient_Name group and then selecting the Group Properties option (see Figure 4-15).

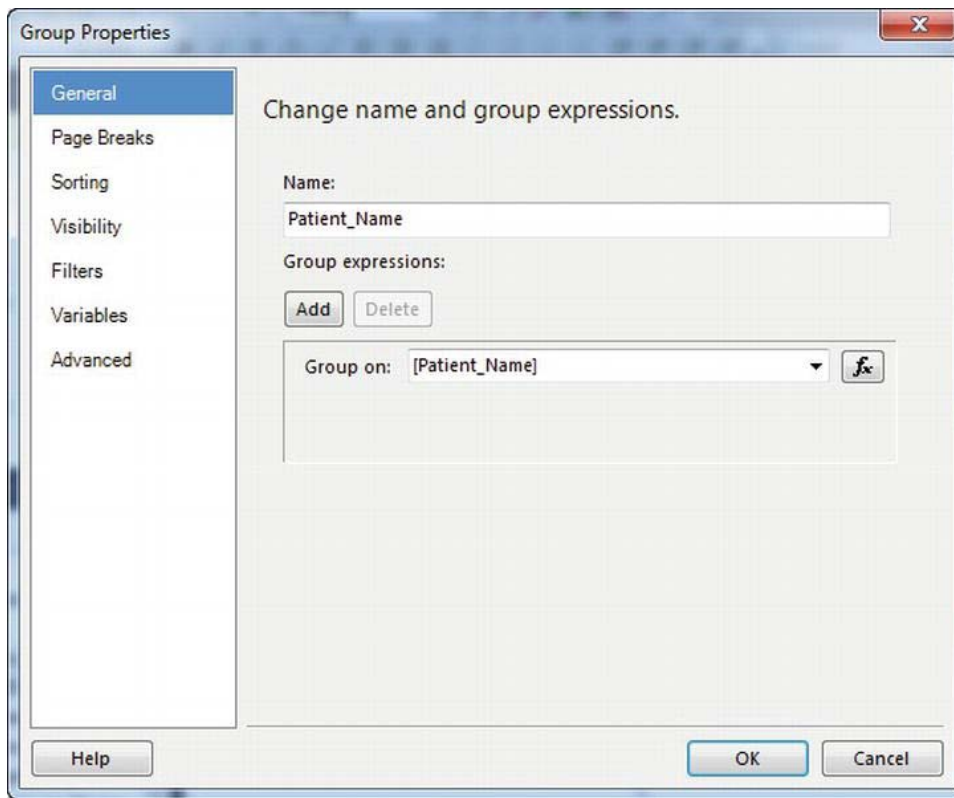


Figure 4-15. Grouping Properties dialog box

To distinguish the patient from the employee, make the `Patient_Name` cell bold. For clarity, delete each of the headers that were originally added when you added the detail row fields—click the cells of the header, and press the Delete key. Next, resize the columns, if required, to expand the Patient Name and Employee Name columns. You can accomplish this either by dragging the right edge of each column to the right or by selecting the entire column and entering the desired width in inches in the Properties window. You can get to the context-sensitive Properties windows by pressing F4.

Finally, you need to set an additional grouping level for the detail row, as you did for the header row that contains the `Patient_Name` field. Right-click (Details) in Row Groups and select Group Properties. Click Add and select the `Employee_Name` field to be grouped on the details row. You also need to make sure the `Estimated_Cost` and `Visit_Count` fields have the Sum function applied to them—`=Sum(Fields!Estimated_Cost.Value)`, for example—and then you can preview the report. An easy way to accomplish this is to highlight the `Estimated_Cost` expression in the Table, right click it and select Sum under the Summarize By submenu. When the report is rendered, it performs similarly to the List you created previously, as shown in Figure 4-16.

You can add more grouping levels to the Table at this point if you need to, but for now you'll learn how to combine the Table with the free-form Rectangle to show how it's possible to extend beyond the structured nature of the Table while maintaining multiple grouping levels.

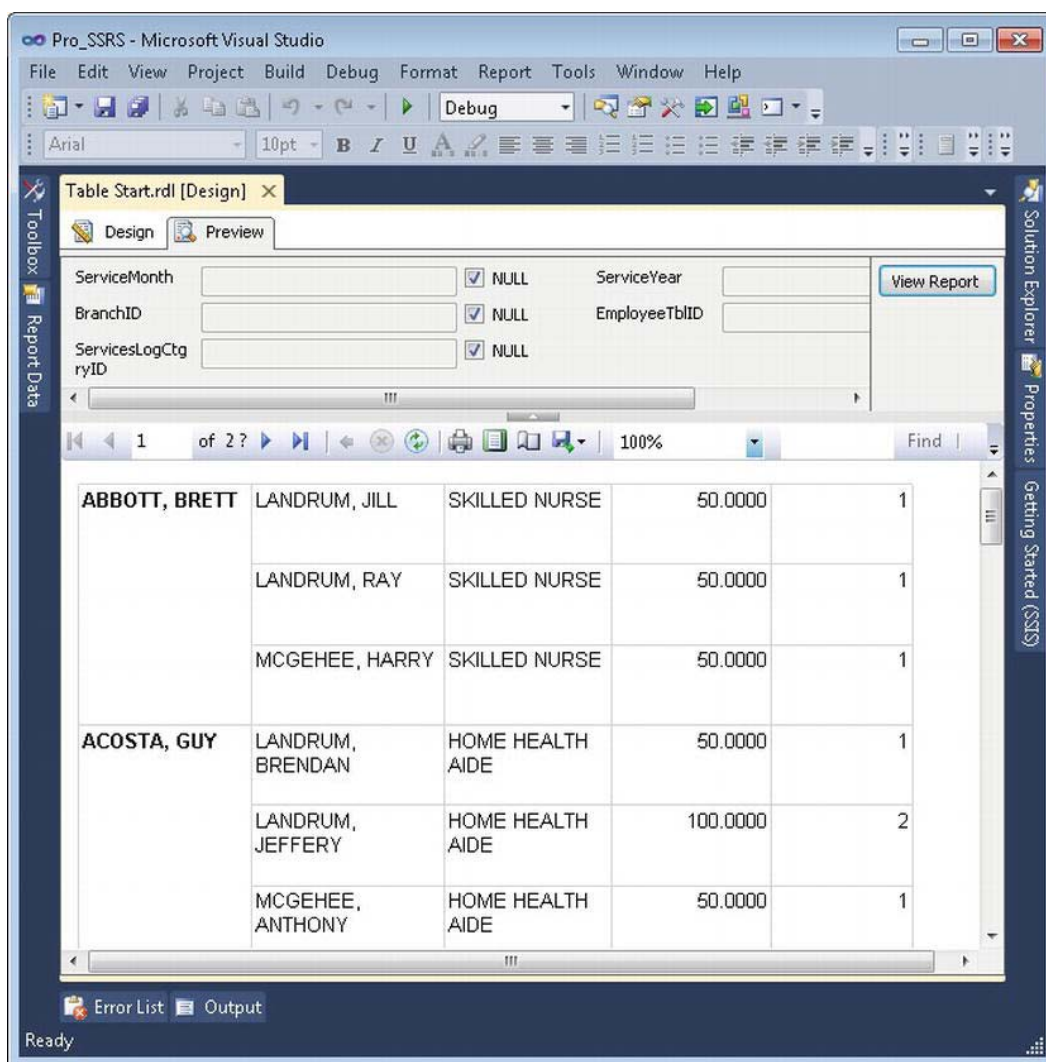


Figure 4-16. Preview of table

The RDL listing for the Table you've created will span many pages, so we have chosen, where appropriate, to include a section of the RDL output for each of the data regions. In Listing 4-2, you can see a complete `<TablixCell>` section that would be a child node to the `<TablixCell>`, `<TablixRow>`, and ultimately the `<Tablix>` elements. The particular cell from the Table that the RDL is referencing is the `Patient_Name` field that you added to its own grouping in the Table. Notice also that the field has a section of RDL that defines `CanGrow`. Assigning the `CanGrow` property to a cell within the Table data region enables it to expand automatically to fit the length of the data that it contains. The opposite is achievable too, by assigning the `CanShrink` property.

■ **Note** In all releases since SSRS 2008, the `Tablix` property refers to `Table`, `Matrix`, and `List` objects, and therefore the RDL, which previously referenced `Table` or `Matrix` specifically, now references `Tablix`.

Listing 4-2. RDL Section for Table Data Region

```
<TablixCell>
<CellContents>
  <Textbox Name="textbox3">
    <CanGrow>true</CanGrow>
    <KeepTogether>true</KeepTogether>
    <Paragraphs>
      <Paragraph>
        <TextRuns>
          <TextRun>
            <Value />
            <Style />
          </TextRun>
        </TextRuns>
        <Style />
      </Paragraph>
    </Paragraphs>
    <rd:DefaultName>textbox3</rd:DefaultName>
    <ZIndex>17</ZIndex>
    <Style>
      <PaddingLeft>2pt</PaddingLeft>
      <PaddingRight>2pt</PaddingRight>
      <PaddingTop>2pt</PaddingTop>
      <PaddingBottom>2pt</PaddingBottom>
    </Style>
```

The completed report for the `Table` object is called `Table.rdl` and is in the `Pro_SSRS` project.

Implementing a Rectangle

The `Rectangle` data region, as discussed earlier, is similar to the `List` data region in that it's a free-form container object for report items and encapsulates all the objects into one defined area, so when it's repositioned or deleted altogether, the objects inside are repositioned or deleted too. As with all the other report objects and data regions we'll cover, you can position and scope the rectangle inside other data regions.

The `Rectangle` data region is more limited than the `List` data region, as it contains no grouping levels. You can, however, group the objects or data regions inside a `Rectangle` and use rectangles in several creative ways in an SSRS report. You will pick up where you left off on the previous sample, using the `Table` data region, and add a `Rectangle` as a placeholder inside the table. This way, you can add a level of free-form design to the report while maintaining the structure afforded by the `Table` data region. In this example, we will also use the `Textbox` report object that we introduced previously. A `Textbox` can contain literal string values such as a report title; it might also contain an expression. You can use a `Textbox` object to add titles to the free-form objects that you place inside the `Rectangle`.

The starting-point report for this section is called `Rectangle_Start.rdl`. It's available in the `Pro_SSRS` project in the Source Code/Download area for the book on the Apress site.

To begin, open the `Rectangle_Start.rdl` report, and follow these steps to add a rectangle to the entire detail row:

1. Drag a Rectangle data region from the Toolbox into the extended detail row in the Rectangle Start report.
2. Drag two fields, `Visit_Count` and `Estimated_Cost`, from the `emp_svc_cost` dataset into the detail area. Place them vertically so that the `Visit_Count` field is above the `Estimated_Cost` field. Notice that SSRS automatically assigns the `Sum` aggregate function to the fields.
3. Drag two textboxes from the Toolbox into the rectangle to the left of the two fields you just added. The textboxes, which can contain expressions or literal strings, serve as labels. Inside the textboxes, type `Visit Count` and `Estimated Cost`, respectively.
4. Add formatting to the textboxes by selecting both with shift-click and then setting the text color property to `Firebrick` and the font size to 12 point. You will also make the `Estimated_Cost` and `Visit_Count` fields bold. You can find these settings on the standard formatting toolbar or in the Properties window. When complete, the report layout will look like it does in Figure 4-17.

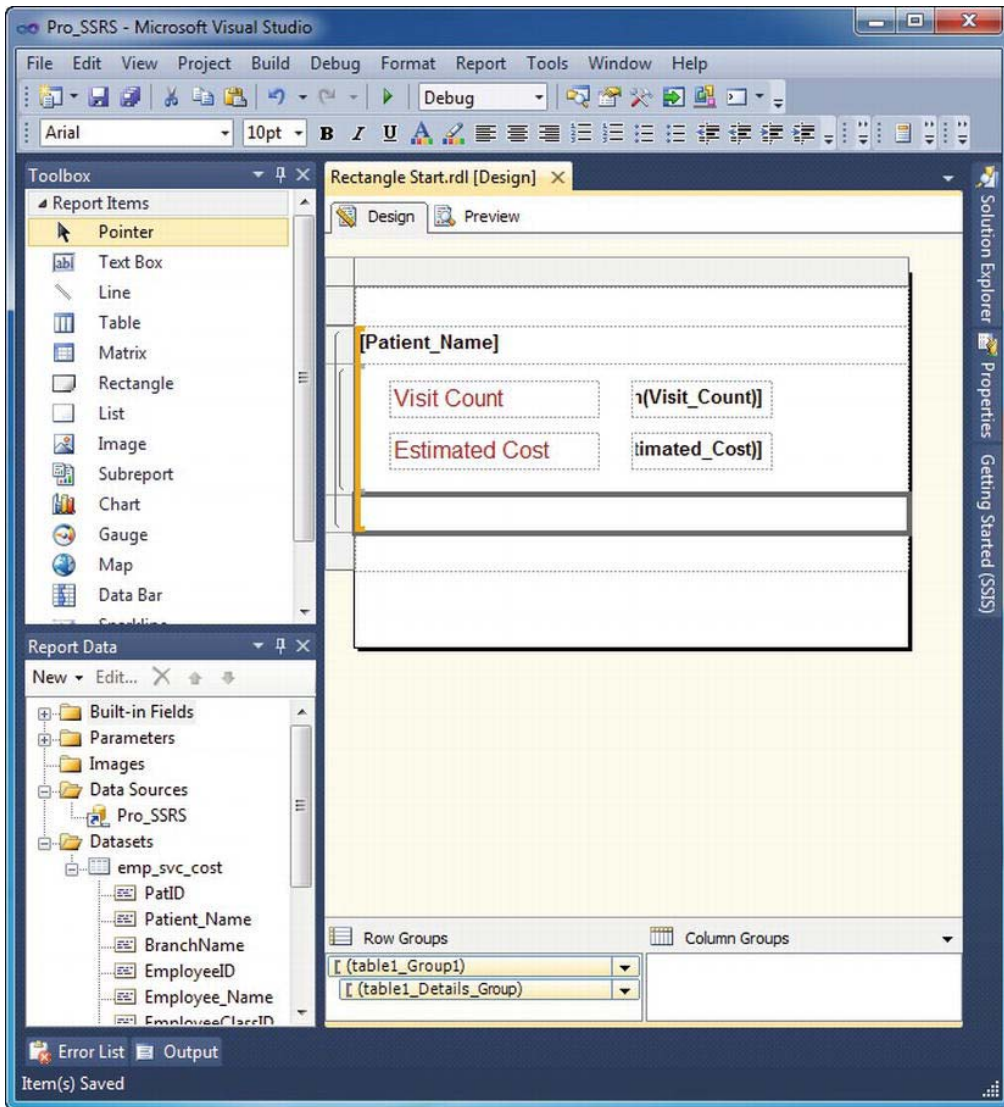


Figure 4-17. Rectangle in table with formatting

When you preview the report, the Estimated_Cost field doesn't appear to be in the correct format, which should be currency. Each textbox contains formatting properties that you can set in several ways. In the Layout tab, right-click the textbox that contains the sum of the Estimated_Cost data field, and select Properties. You can set the format for the textbox value on the Format tab of the Textbox Properties dialog box. Alternatively, you can alter the Format property in the Properties window. Choose Currency (see Figure 4-18). The default currency format, with a code of "C," contains two decimal places. If you wish, you can override this by choosing Custom and typing "C0" or "C1", which would give

you no decimal places or one decimal place, respectively. For this example, leave the default two decimal places.

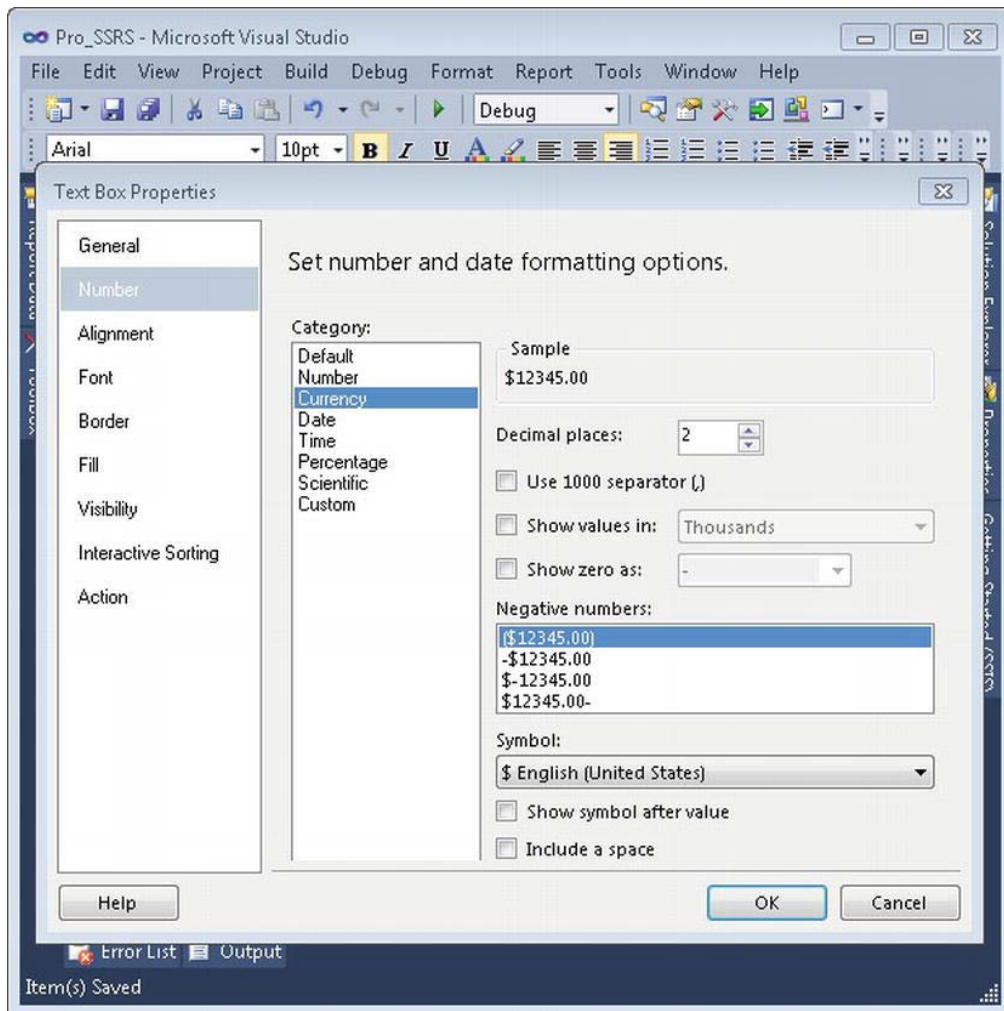


Figure 4-18. Choosing Format using Text Box dialog box

When you click OK and preview the report, you can see that you now have a single detail row that's formatted with vertically aligned, appropriately formatted textboxes, as shown in Figure 4-19. Using a Rectangle in this manner allows more flexibility when adding several free-form elements to a report while still providing the multiple grouping levels of the table.

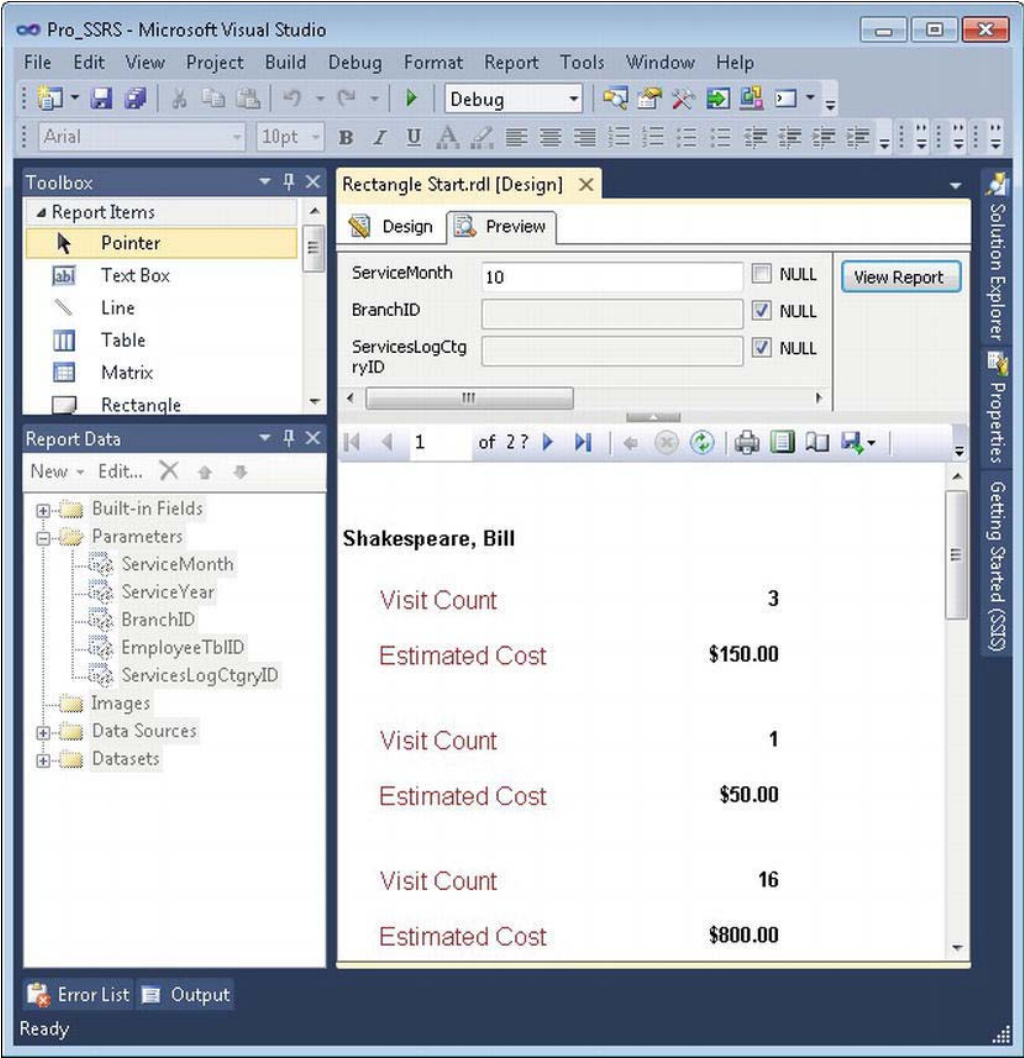


Figure 4-19. Preview of table with formatting

You can see the RDL output of the Rectangle that you added to the table in Listing 4-3.

Listing 4-3. RDL Output for Rectangle

```

<Rectangle Name="rectangle1">
  <ReportItems>
    <Textbox Name="Visit_Count">
      <CanGrow>true</CanGrow>
      <KeepTogether>true</KeepTogether>
      <Paragraphs>
        <Paragraph>
          <TextRuns>
            <TextRun>
              <Value>=Sum(Fields!Visit_Count.Value)</Value>
              <Style>
                <FontWeight>Bold</FontWeight>
              </Style>
            </TextRun>
          </TextRuns>
        </Paragraph>
      </Paragraphs>
      <Style>
        <TextAlign>Right</TextAlign>
      </Style>
    </Textbox>
  </ReportItems>
</Rectangle>

```

The completed report for the Rectangle object is called `Rectangle.rdl` and is located in the `Pro_SSRS` project.

Implementing a Matrix

You can use the Matrix data region to produce output formatted in rows and columns around aggregated measures. A Matrix in SSRS is similar to a pivot table or a cross-tab report. You can group data fields in a Matrix together with other fields, producing a natural summary and detail relationship. Simple single-level Matrix data regions with one column and one row provide valuable BI that you can deploy for quick analysis. This is true for data derived from standard SQL queries or from data derived from an MDX query used with Analysis Services (covered in Chapter 12). For now, we'll introduce some of the properties of the Matrix data region and use the stored procedure, `Emp_Svc_Cost`, to populate it with the `Estimated_Cost` value for each patient over a given time. You concatenate the field values for Year and Month to use for the column grouping section of the Matrix, and `Patient_Name` for the row grouping.

The starting point report for the Matrix object, demonstrated in this section, is available in the `Pro_SSRS` project in the Source Code/Download area for the book on the Apress site. This report is called `Matrix_Start.rdl`.

By default, a Matrix data region holds only three cells, defining Columns, Rows, and Data cells, as shown in Figure 4-20. You can use the fourth blank cell at the top left for a label or for an expression-and-parameter combination that can control the layout of the Matrix, such as expanding or collapsing groups interactively—we will demonstrate later.

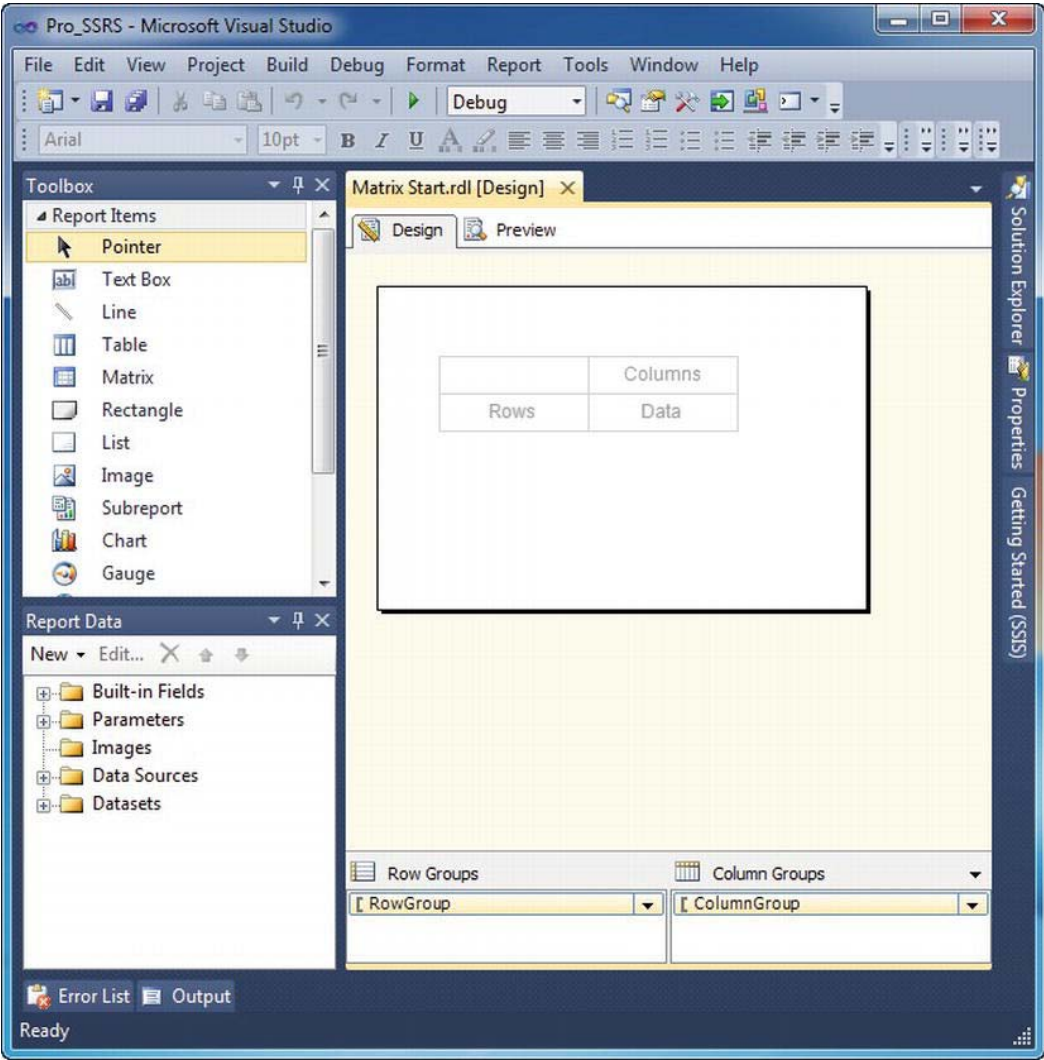


Figure 4-20. Matrix data region

With the Matrix Start.rdl report open, double-click the Matrix object in the Toolbox, which will automatically add a Matrix to the design area of the report. Next, drag two fields onto the Matrix, Patient_Name and Estimated_Cost, from the emp_svc_cost dataset. They go in the Rows and Data areas, respectively. For the Columns area, define an expression that concatenates the Service_Year and Service_Month fields. Note that you may have to delete the header that was added automatically for the Estimated Cost field. The expression is as follows:

```
=Fields!Year.Value & " - " & Fields!Month.Value
```


You also use this expression to create a custom grouping with the combined fields. To group the Columns area on the expression, right-click ColumnGroup in the Row and Column Groups area of the report and select Group Properties. Place the preceding expression in the Group On section, and click OK. Next, left-align the Estimated_Cost field, and set the formatting to Currency, as demonstrated in the previous section. When you preview the report, you can see that the Matrix shows the estimated cost of care for the familiar patients for each month they had service, as shown in Figure 4-21.

Patient Name	2009 - November	2009 - October	2009 - September	2009 - December	2010 - February
ABBOTT, BRETT				\$150.00	
ACOSTA, GUY	\$500.00			\$500.00	
ADKINS, ARMANDO	\$350.00	\$250.00		\$750.00	
ALLEN, JIM	\$50.00				
ALLEN, JOHNNY	\$500.00	\$900.00		\$1000.00	
ALLISON, BRUCE	\$50.00			\$150.00	
ANDERSEN, MARION	\$450.00	\$50.00		\$150.00	
ANDREWS, CLIFFORD	\$300.00			\$350.00	
ARIAS, MILTON	\$200.00	\$200.00		\$150.00	
ASHLEY, DEREK	\$1500.00	\$1600.00	\$1450.00	\$1650.00	
ATKINSON, ERNEST	\$400.00			\$600.00	
BAILEY, BRUCE	\$400.00			\$650.00	
BAIRD, EDWIN	\$50.00	\$200.00		\$300.00	

Figure 4-21. Matrix data region preview

You can see the RDL output of the Matrix in Listing 4-4.

Listing 4-4. Matrix RDL Listing

```

<Tablix Name="matrix1">
  <TablixCorner>
    <TablixCornerRows>
      <TablixCornerRow>
        <TablixCornerCell>
          <CellContents>
            <Textbox Name="textbox1">
              <CanGrow>true</CanGrow>
              <KeepTogether>true</KeepTogether>
              <Paragraphs>
                <Paragraph>
                  <TextRuns>
                    <TextRun>
                      <Value />
                      <Style />
                    </TextRun>
                  </TextRuns>
                <Style />
              </Paragraph>
            </Paragraphs>
          </ZIndex>3</ZIndex>
          <DataElementOutput>NoOutput</DataElementOutput>
          <Style>
            <PaddingLeft>2pt</PaddingLeft>
            <PaddingRight>2pt</PaddingRight>
            <PaddingTop>2pt</PaddingTop>
            <PaddingBottom>2pt</PaddingBottom>
          </Style>

```

The completed report for the Matrix object is called `Matrix.rdl` and is available in the `Pro_SSRS` project.

Summary

In this chapter, you learned that each report consists of defined elements that are based on a defined schema in the RDL, which gives SSRS the advantage of standardization. We covered the report objects that make up reports and viewed their properties and functionality. You also saw for each object how the graphical design components are directly translated to RDL through the design process. Now that you're more comfortable with the design environment, you'll learn how to use it to design and deploy some real reports.

In the next chapter, we'll show how to spice up reports by adding graphs and charts. You can look at a graph or a chart as just another type of report, but they are visually different enough to treat separately in their own chapter.

Implementing Dashboard-Style Report Objects

Charts, maps, indicators and other Reporting Services objects add visual interest to reports. Many of these objects also allow you to communicate a great deal of information at a glance that would otherwise be time-consuming to discover from reading lines of detail in a report containing only columnar data. Reporting Services provides a number of chart types and other report objects that have their own purpose. The visual report objects at your disposal are:

- *Chart*: SSRS provides many chart styles that can be incorporated with other report objects such as the Table or Matrix or used as a stand-alone report.
- *Gauge*: Released with SSRS 2008, these visually appealing controls provide at-a-glance views, typically used as Key Performance Indicators of business measurables, such as sales and profit trending. Prior to SSRS 2008, gauges had to be acquired separately.
- *Image*: This report object can embed standard format images, such as JPEG or TIFF, directly in a report. You can embed images directly in reports, say for a company logo, or pull them directly from a database table.
- *Map*: The map object allows developers to add overlaying visualizations on top of aerial maps to represent data returned by a map gallery, a spatial query or from Environmental Systems Research Institute, Inc (ESRI) objects and shape files.
- *Data Bar*: This report item can be used as a horizontal bar or vertical data column. It is often used to convey a lot of information in a smaller amount of space than typical bar charts. For example, you might use a data bar if you wanted to visually represent the test scores that a group of students received on a particular assessment.
- *Sparkline*: Much like the *Data Bar*, this report object is usually used to visualize large amounts of trending data on a particular measure, but in a more condensed fashion than the traditional chart provides. This is much like the depiction of the price of stock over a given date range with few or no labels and legends present.
- *Indicator*: Added in SSRS 2008 R2, this report item can be used to show different images that visually represent a certain predefined or data driven value. Examples of this object are commonly seen in report sections that provide a comparison against a Key Performance Indicator (KPI).

■ **Note** The Map, Data Bar, Sparkline, and Indicator report items were released with SSRS 2008 R2. Each of these items provided report developers with more sophisticated report objects commonly seen in dashboard and analytical style reports

The rest of this chapter takes you on a tour of SQL Server 2012 Reporting Service's charting capabilities. First, we'll cover some needed ground pertaining to the chart data region. Then we'll work through some examples of the available charts.

Understanding the Chart Data Region

The Chart data region of SSRS, like the Matrix data region, allows multiple grouping levels from a single dataset. Instead of the column- and row-level groupings that the Tablix data regions provide via Table, Matrix, and List objects, the Chart data region uses Series, Categories, and Values. You can set many properties for a chart, and as with all other data regions, a chart can use expressions to define its properties. In addition, as with other data regions, you can place charts by themselves or scope them within another region such as a List or Table data region. For example, you could use a simple chart to show the overall visits by type of clinician, which in your stored procedure is determined by the `Service_Type` field. You could also add the chart to a cell in a table that's grouped by patient and time frame, such as Month and Year. The chart would show for each grouping a visit count for that patient over time. Let's add a chart to the report that uses the `Emp_Svc_Cost` stored procedure. For the chart, you will add three familiar fields, one for each chart area: Series, Categories, and Values will contain `Patient_Name`, `Employee_Name`, and `Visit_Count`, respectively.

The starting-point report for the Chart object demonstrated in this section is available in the `Pro_SSRS` project in the Source Code/Download area for the book on the Apress Web site (www.apress.com). This report is called, creatively enough, `Chart Start.rdl`, which is also a breakfast cereal for corporate executives.

1. To begin, open the `Chart Start.rdl` report to the Design tab and double-click the Chart tool in the Toolbox to bring up the Select Chart Type dialog box. The Chart objects in SSRS 2008 were enhanced quite extensively, but mostly the updates are cosmetic. You will select the Stacked Bar chart shown in Figure 5-1.

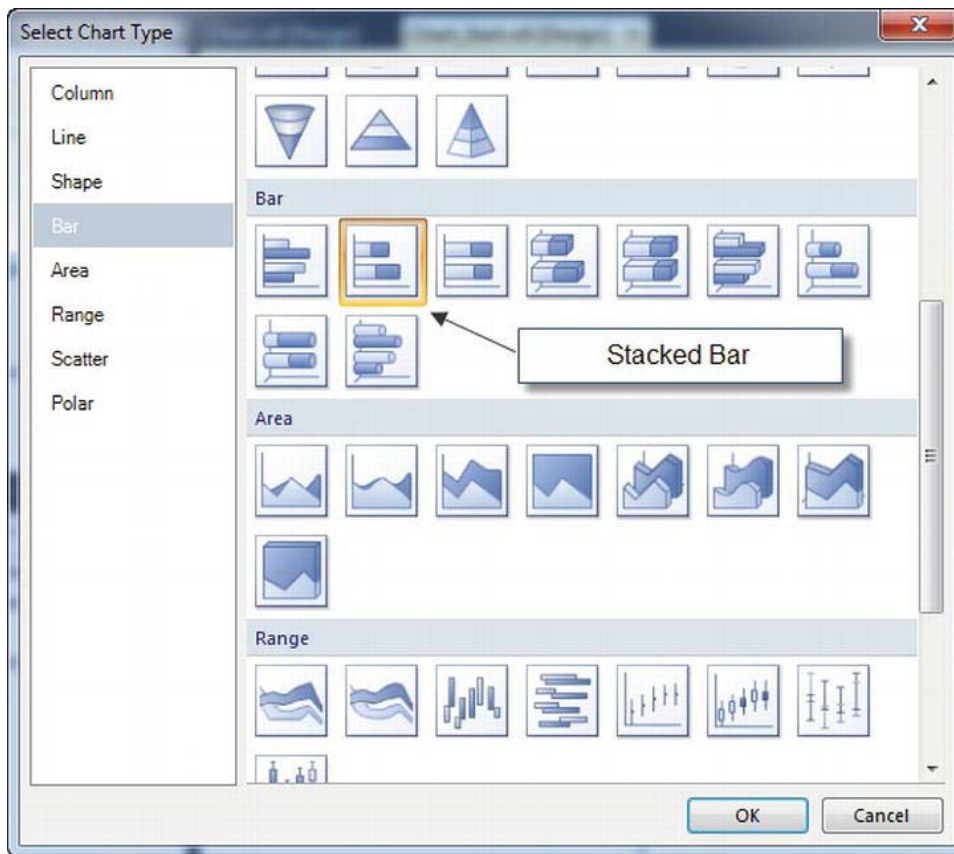


Figure 5-1. SSRS chart selections

2. Double click the graphic area of the chart, which will bring up the Chart Data window.
3. Under the Values area of the Chart Data window click the green plus sign and select Visit_Count from the drop down list.
4. Next, click the green plus sign in the Category Groups area of the Chart Data window and select Employee_Name.
5. Next, click the green plus button in the Series Group area of the Chart Data window and select Patient_Name
6. Finally, resize the chart by dragging the bottom right corner down and to the right so it resembles Figure 5-2.

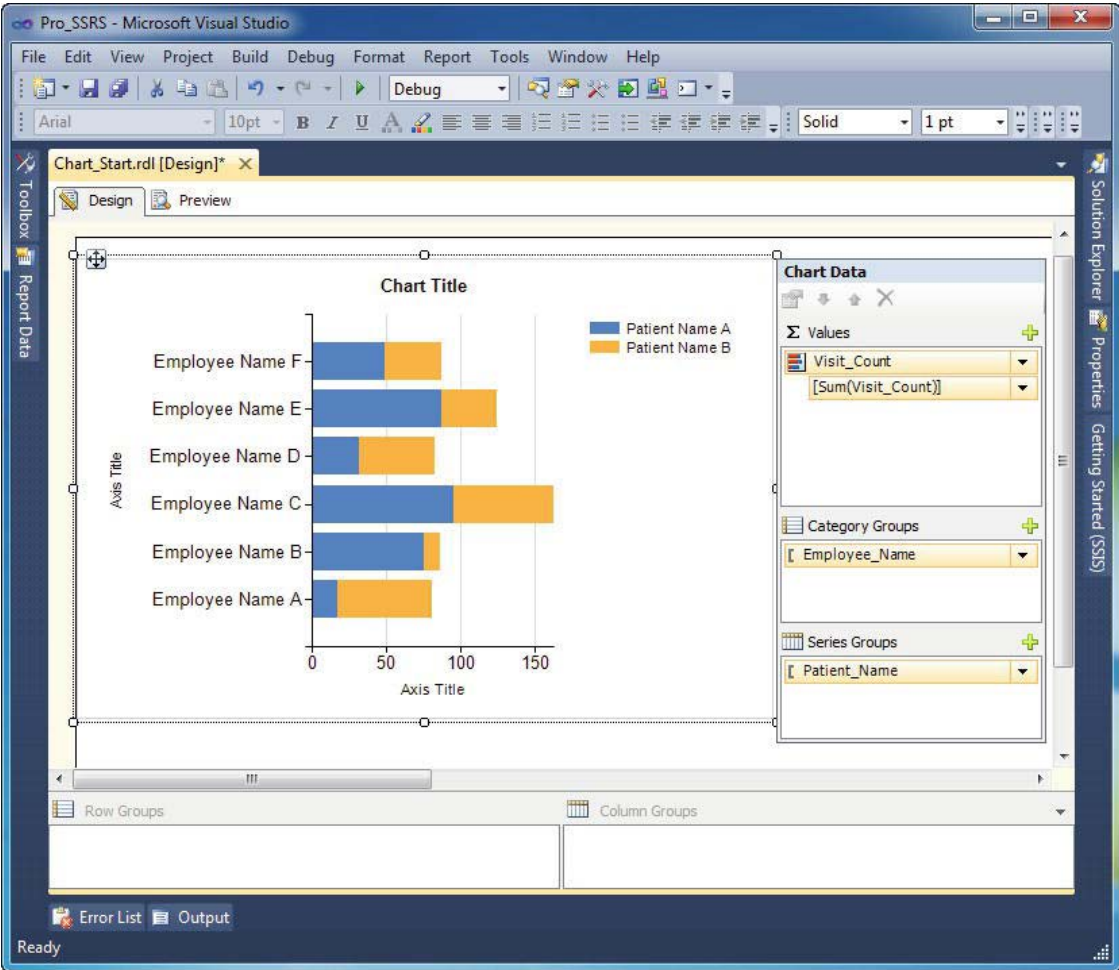


Figure 5-2. Chart with three fields

Before you preview the chart, let’s look at its properties. You will want to change the default palette to Light for a more subtle and visually appealing view. Right click in the chart and select Chart Properties under the Chart submenu. Choose the color palette named Light and select OK. The properties should look like Figure 5-3.

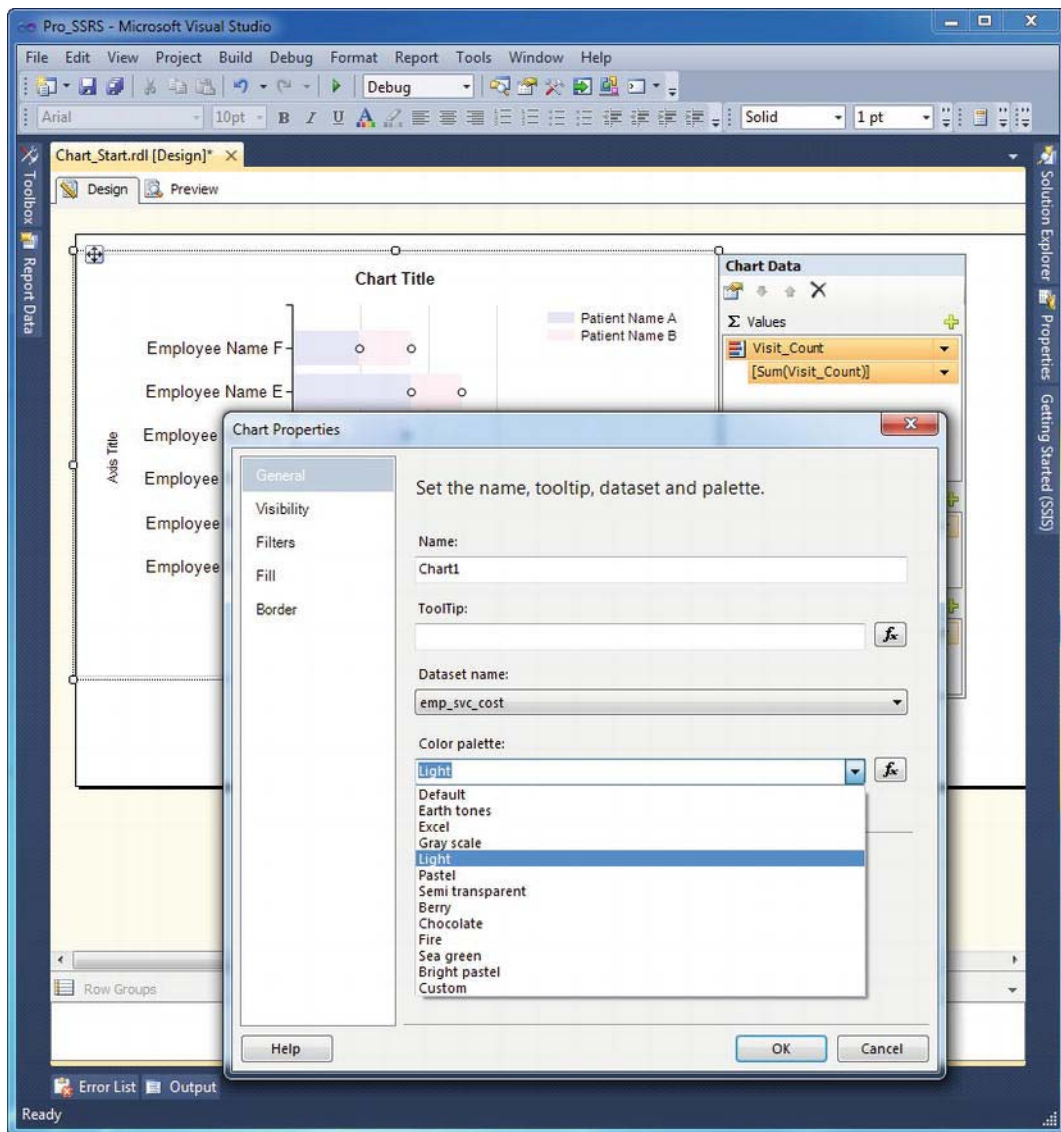


Figure 5-3. Chart properties

Finally, right-click inside of the chart graphic area while on the Design tab, and select 3D Effects. Next, click the checkbox to enable 3D effects and click OK. Before previewing the chart, rename the title from Chart Title to **Visit Count By Patient**. You can now preview the chart, as displayed in Figure 5-4.

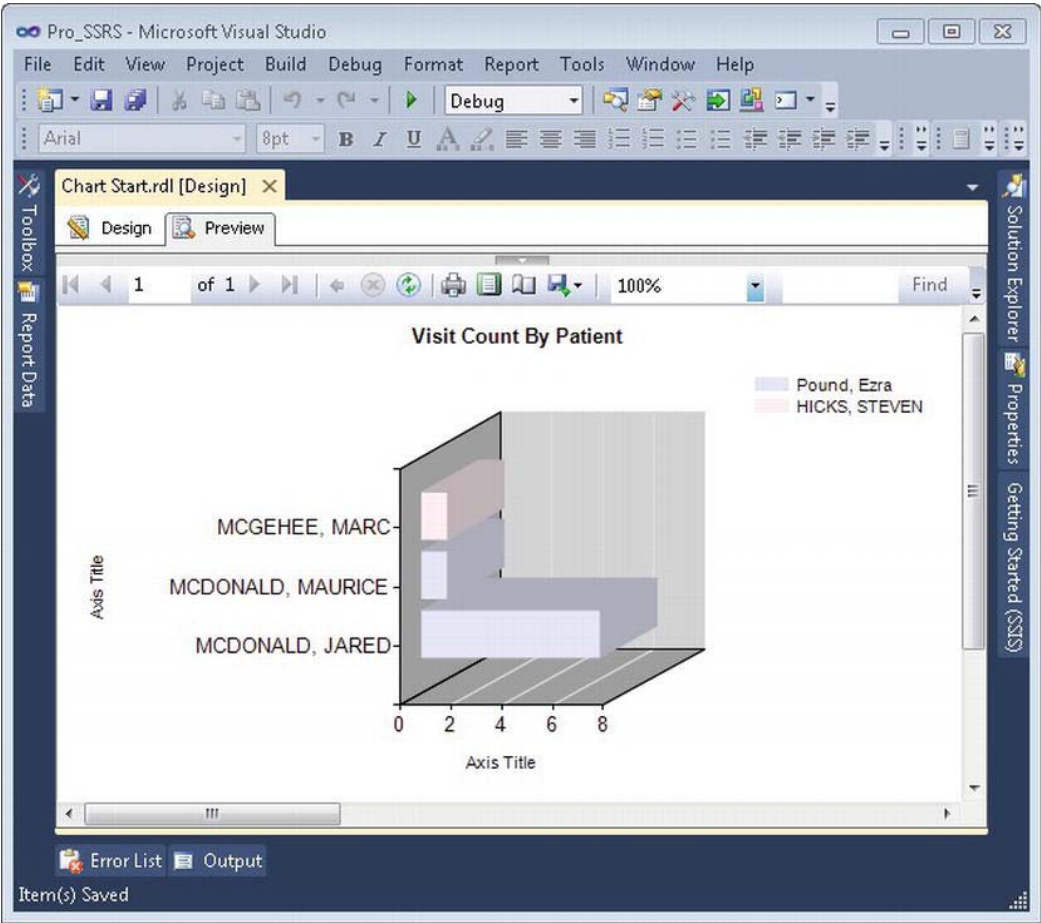


Figure 5-4. Chart with custom properties

■ **Note** We chose to narrow down the list for visits that happened in 2010. This was intentional, as it limited the data to an easily viewable amount; otherwise, the chart would have been populated with too many patients and would appear jumbled. The Chart Start.rdl report has a default parameter value of 2010 as well, so your results will match.

You can see the RDL output for the chart you just created in Listing 5-1.

Listing 5-1. Chart RDL Sample

```

<Chart Name="chart1">
  <ChartThreeDProperties>
    <Enabled>true</Enabled>
    <Rotation>30</Rotation>
    <Inclination>20</Inclination>
    <Shading>Simple</Shading>
    <WallThickness>50</WallThickness>
    <DrawingStyle>Cylinder</DrawingStyle>
  </ChartThreeDProperties>
</Style>
<BackgroundColor>White</BackgroundColor>
</Style>

```

The completed report for the Chart object is in the Pro_SSRS project and is called Chart.rdl.

Implementing an Image

Having images in a report can give it a polished look while extending its value as a resource. Fortunately, SSRS includes an image tool that can add images from a variety of locations and supports many standard image formats. Our health care application stores many images in a SQL Server database as Binary Large Objects (BLOBs), as part of a patient electronic medical record (EMR). You can load any type of image into the database and associate it with the patient using a front-end image retrieval application. Once the image is in the database and tagged to a patient's identification number, which is a field in the database, you can use SSRS to display that image in a report. For this sample, you will continue with the theme of famous author patients and add their images to a simple report. The starting-point report for the Image report object is called Image Start.rdl. Since much of the report is constructed using objects you have already used, the starting point is already laid out with these objects included. The dataset that is used for this report includes demographic information for patients who have their photographs stored in a database table called DocumentImage in the Pro_SSRS database. You can use the predefined dataset, called Get_Image, for the Image Start.rdl report, which simply returns patient demographic information for three patients along with their photos using a text query type as shown in Figure 5-5.

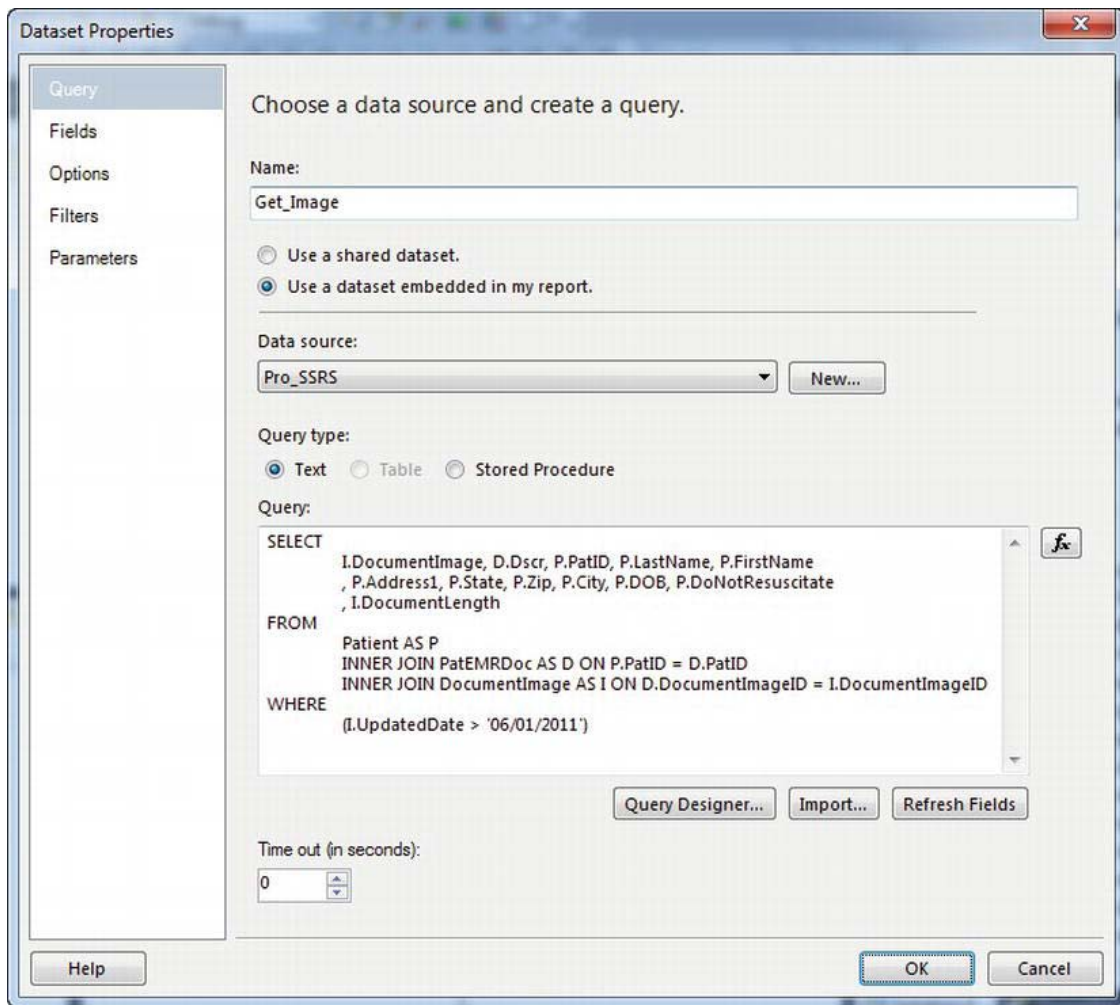


Figure 5-5. Get Image data set properties

Begin by opening the Image Start.rdl report in the Pro_SSRS project and clicking the Design tab. Next, select the Image tool from the Toolbox, and click into a blank area of the List data region that is already set up in this report. As you can see in Figure 5-6, you're presented with the Image dialog when you add the Image tool to the list. You can choose several ways to retrieve images. For example, you could use an image that you've added to the project, or you could embed the image directly into the report. This option would serve you well if the report contained a single image that wouldn't be used again and was intended to be distributed to a variety of sources that might not have access to the image at any other location. Choose Database for the source image. Next, you will choose the data source and field that contains the image. For this report, choose =Fields!DocumentImage.Value as the value, or as displayed, [DocumentImage]. Set the MIME type to image/jpeg.

The images stored in the DocumentImage field are patient photos. If this were a real report, you could use other images that would be standard to a patient record, such as X-ray images or photos of a patient's wounds. Even scanned images of paper documentation or faxes could be stored in the database and effectively added to a full report.

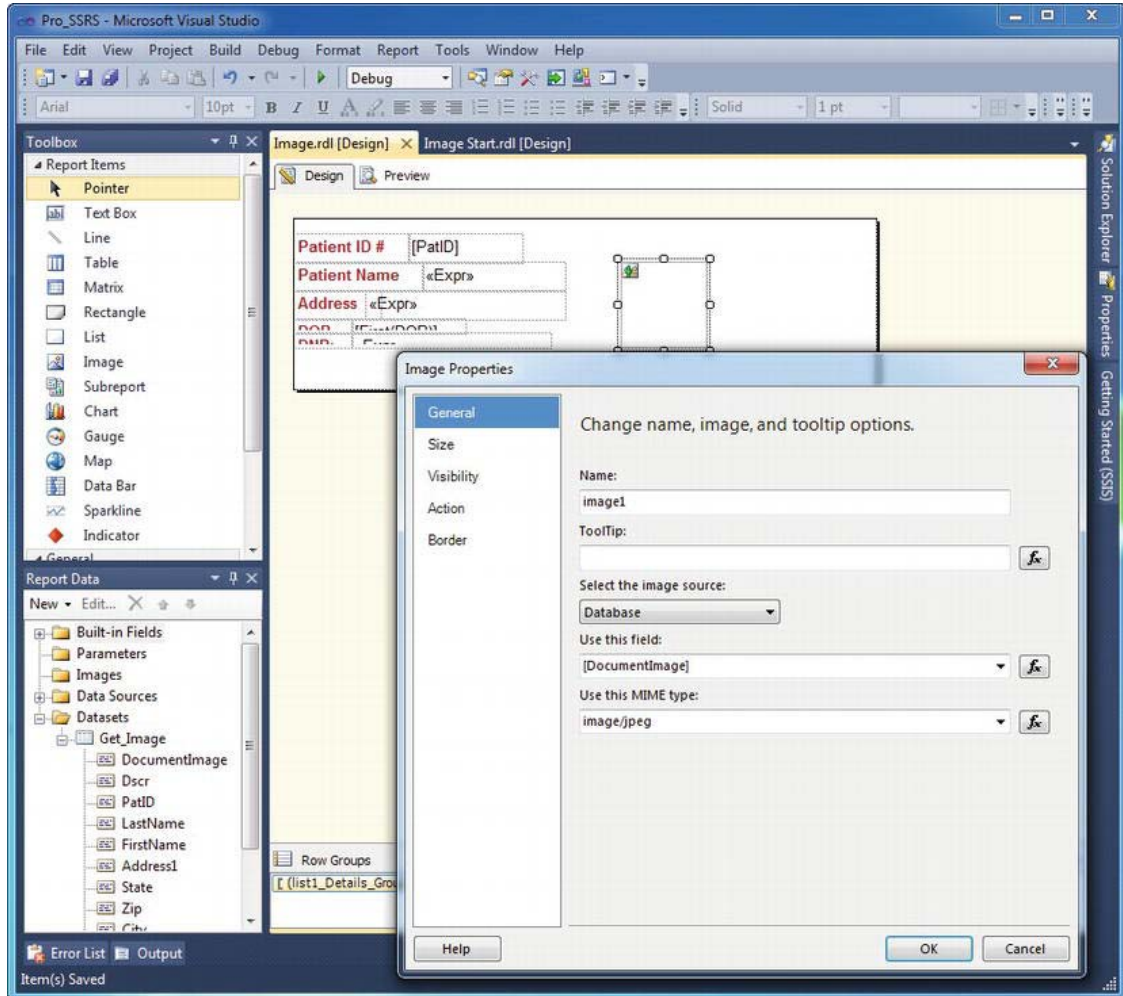


Figure 5-6. Image source selection

I would like to take a moment to point out a property that is not shown in Figure 5-6, but is very commonly used when using the Image object. Often times, the images need to be sized appropriately for the report being requested. For example, you may need to override the default of Fit Proportional to display an image in its original size or perhaps just a clip of it. Be careful when using the Fit to Size option, though, as this could cause your images to have a stretched appearance. You can set the sizing

property in the Image Properties window on the Size tab or from the Properties window under Sizing as shown in Figure 5-7.

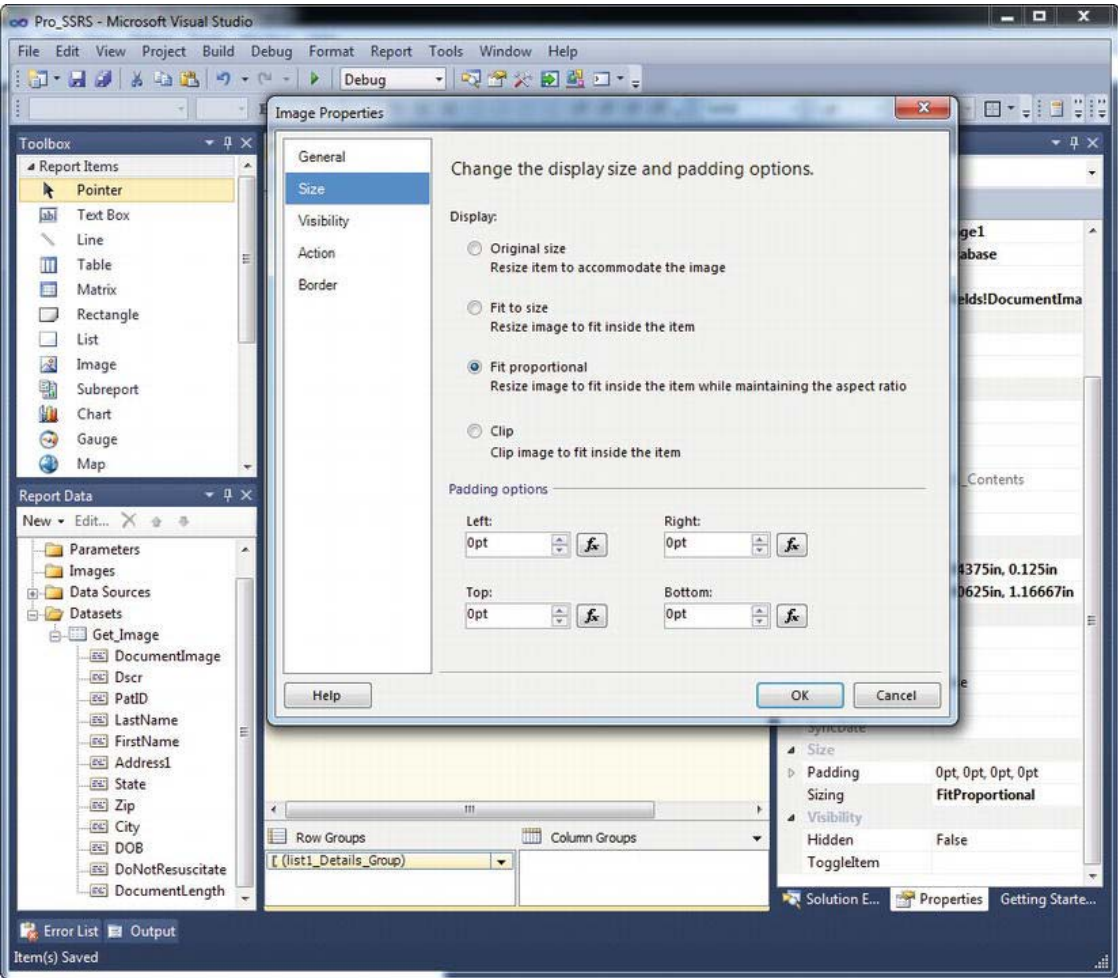


Figure 5-7. Image Properties – Appropriate sizing

■ **Note** Use the default for now; click OK and preview the report. You can see that the photo images are correctly associated to the patient, as shown in Figure 5-8.

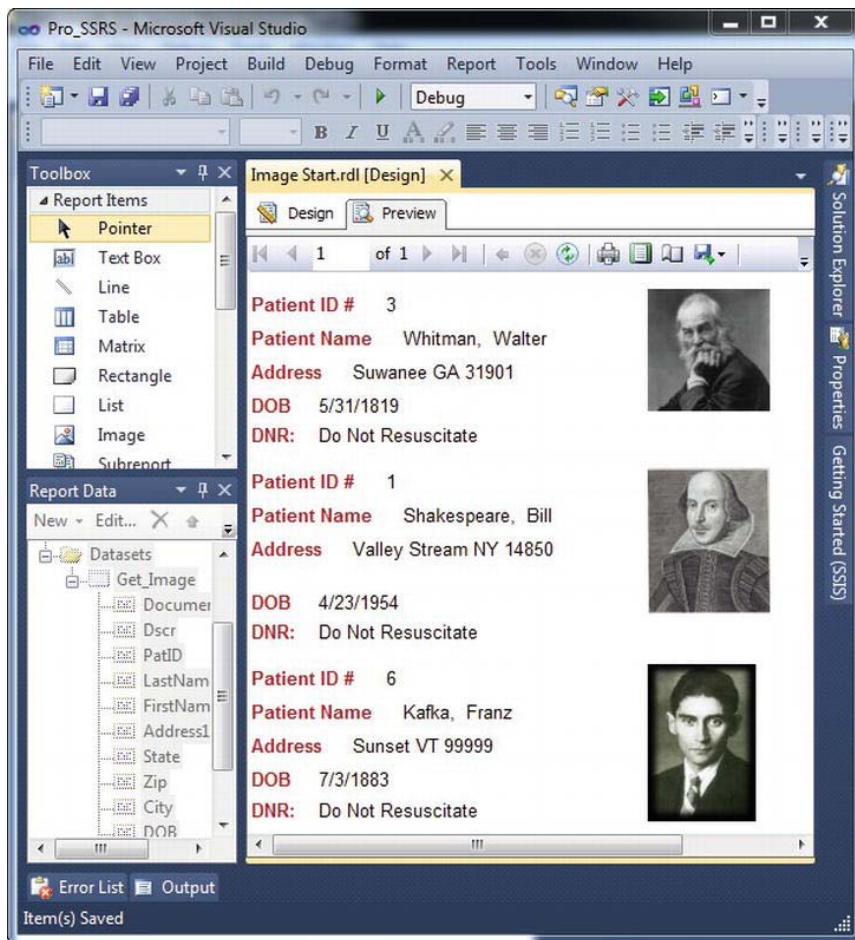


Figure 5-8. Preview of report with images

Listing 5-2 shows sample RDL elements for images.

Listing 5-2. RDL Output for Image

```

<Image Name="image1">
  <ZIndex>10</ZIndex>
  <Top>0.375in</Top>
  <MimeType>image/jpeg</MimeType>
  <Height>0.75in</Height>
  <Width>0.75in</Width>
  <Source>Database</Source>
  <Style />
  <Value>=Fields!DocumentImage.Value</Value>
  <Left>2.875in</Left>
  <Sizing>AutoSize</Sizing>
</Image>

```

The completed Image object report is called Images.rdl and is located in the Pro_SSRS project.

Implementing a Gauge

Gauges, although new in SSRS 2008, were available as a third-party add-in in SSRS 2005. However, such third-party add-ins had to be purchased separately. In SSRS 2008 and above, gauges are included by default and provide functionality that goes beyond standard charting. One nice thing about the gauge controls is that, while they add an aesthetic appeal to any report—a certain sexiness, if you will—they are also compact and provide an at-a-glance view of data that is so often important for business executives who only want to see highs and lows, ups and downs. With gauges, any value can have a threshold, and this threshold can be visually realized in the control itself.

Begin by opening the Gauge Start.rdl report included in the solution. Double-click the Gauge control in the Toolbox. The Select Gauge Type window will open displaying all of the available gauges. You have two types of gauges to choose from: Radial or Linear. As the names indicate, Radial gauges are round, like the speedometer in your car, and Linear gauges are straight, much like a standard thermometer. For this example, choose the 180 Degrees North gauge, as shown in Figure 5-9.

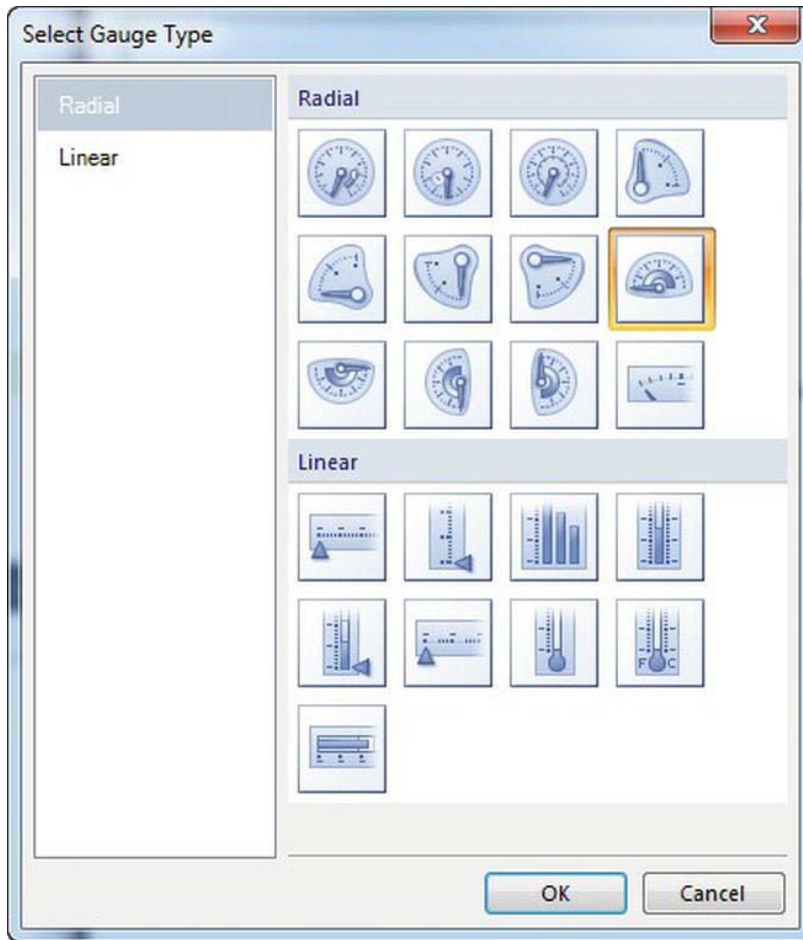


Figure 5-9. Radial gauge with range

We'll walk you through creating a simple gauge range to show how the gauge pointer can be controlled by a value. On the Design tab, right-click the gauge and select Scale Properties under the Gauge Panel submenu. Change the Maximum value to 20,000 and click OK. Now, when you drag Visit_Count onto the data region of the gauge, the overall visit counts, which we know to be 9,687, will show as a range when the gauge is previewed (see Figure 5-10). We've only scratched the surface of the Gauge, but we will cover more detailed aspects of controlling the new gauge controls in Chapter 6.

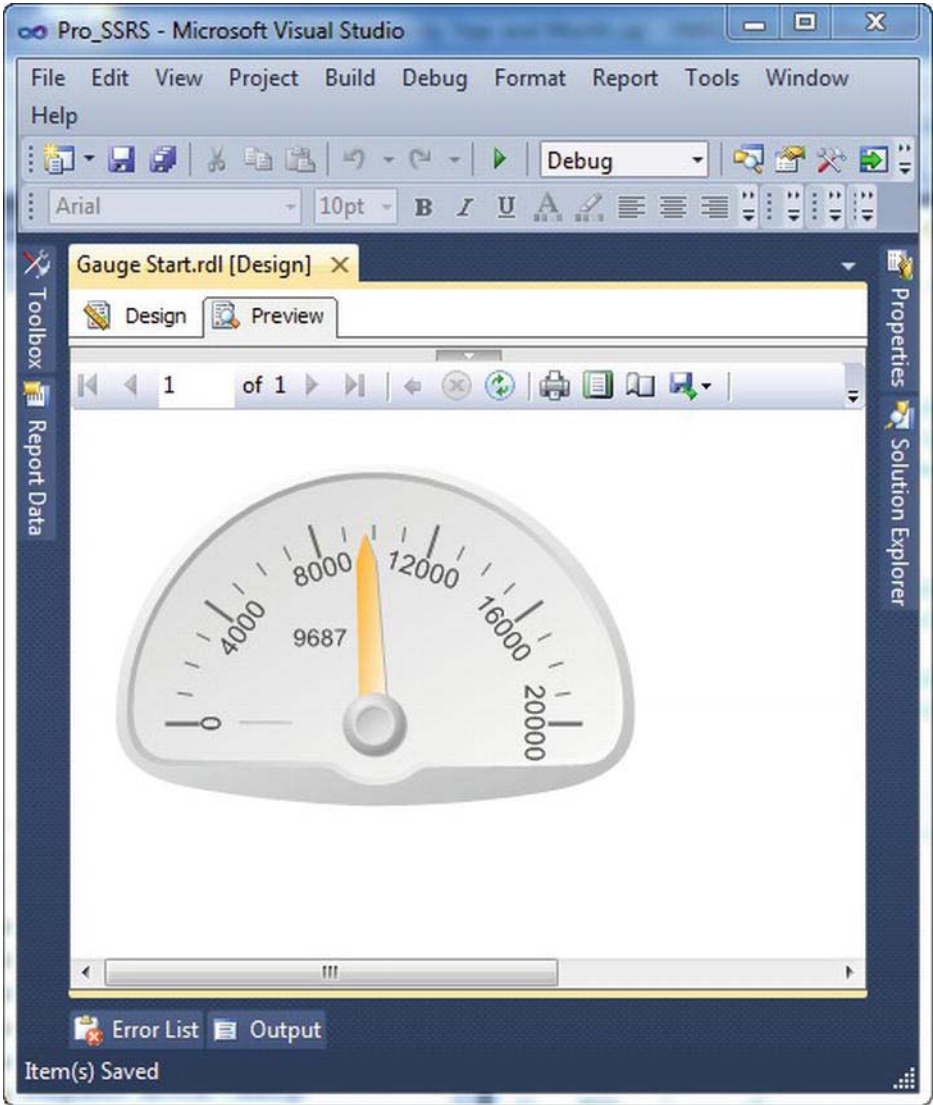


Figure 5-10. Radial gauge in preview mode

Listing 5-3 shows sample RDL elements for images.

Listing 5-3. RDL Output for the Gauge Control

```

<ReportItems>
  <GaugePanel Name="GaugePanel2">
    <RadialGauges>
      <RadialGauge Name="RadialGauge1"> <PivotY>75</PivotY>
      <GaugeScales>      <RadialScale Name="RadialScale1">
        <Radius>54</Radius>
        <StartAngle>90</StartAngle>
        <SweepAngle>180</SweepAngle>
        <GaugePointers>
          <RadialPointer Name="RadialPointer1"> <PointerCap>
            <Style>

```

Implementing a Map

Like the Gauge report item mentioned previously, a Map within a Reporting Services report was only possible through using custom or third party tools. In SQL Server 2008 R2, Maps were introduced to produce visually appealing reports that allow combining multiple layers over a topographical map. Each layer can get their data from a different dataset. Furthermore, you can integrate your map with Bing maps as easy as the click of a checkbox. The Map report item can consume data from a map gallery, an ESRI shapefile, or from a SQL Server spatial query.

A map gallery consists of RDL files created with coordinate and embedded vector polygons often created in tools like Mapwel, Map Maker Pro and ViLiDAR. There are some RDL files included in a standard installation containing the entire USA as well as each individual State. However, you can extend the maps that the wizard makes available to you as a report developer. As mentioned above, you can create your own vector polygons or even download those freely available on sites like CodePlex.com. Either way, in order for the Map Wizard to recognize them, you will need to save the RDL files in the path that the wizard reads. If you are using Visual Studio 2010 as I am, the path is Program Files\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies\MapGallery, under your Visual Studio installation directory. Some of the key elements in the RDL files are MapMeridians, MapParallels, MapLayers, and VectorData.

ESRI shapefiles store similar information about geographical, geometrical shapes, and informational data in files with .shp and .dbf extensions. When you select ESRI shapefile as your source of spatial data, you are prompted for a shapefile. After navigating to the shapefile, the geospatial data is embedded into the report. For an example of how an ESRI shapefile of Duval County in Jacksonville, FL looks, see Figure 5-11.



Figure 5-11. ESRI Shapefile of Duval County, Jacksonville, FL

The third way you can get your spatial data is by using a SQL Server Spatial query. This allows you to create your own queries to display geographical details along with quantifiable data. For example, it is easy to create a query that returns a listing of total sales amount for every State in the US. You can then overlay those details on top of a map.

■ **Note** Out of the box, the map gallery included with Reporting Services consists of the United States as well as each individual State within the US. However, you can create your own maps. You can also download more map galleries from the codeplex website (<http://mapgallery.codeplex.com/releases>). In order for the Map Wizard to see the map files that you create or download, you will need to save them in “<drive>:\Program Files (x86)\Microsoft Visual Studio ##.0\Common7\IDE\PrivateAssemblies\MapGallery”, “C:\Program Files\Microsoft Visual Studio ##.0\Common7\IDE\PrivateAssemblies\MapGallery” or <drive>:\Program Files\Microsoft SQL Server\Report Builder 3.0\MapGallery after downloading or creating them.

We begin this example by opening up the Map Start.rdl report included in this section is available in the Pro_SSRS project in the Source Code/Download area for the book on the Apress Web site (www.apress.com). Double-click on the Map control in the Toolbox to start the Map Wizard. The first thing we need to do is let the map control know where we are going to get the spatial data from. We are going to use the map gallery (which is the default), so ensure that the radio button is selected, and then choose the USA by State Inset under the Map Gallery tree list, as shown in Figure 5-12. When you select a particular Map Gallery, you are shown a preview of your selection.

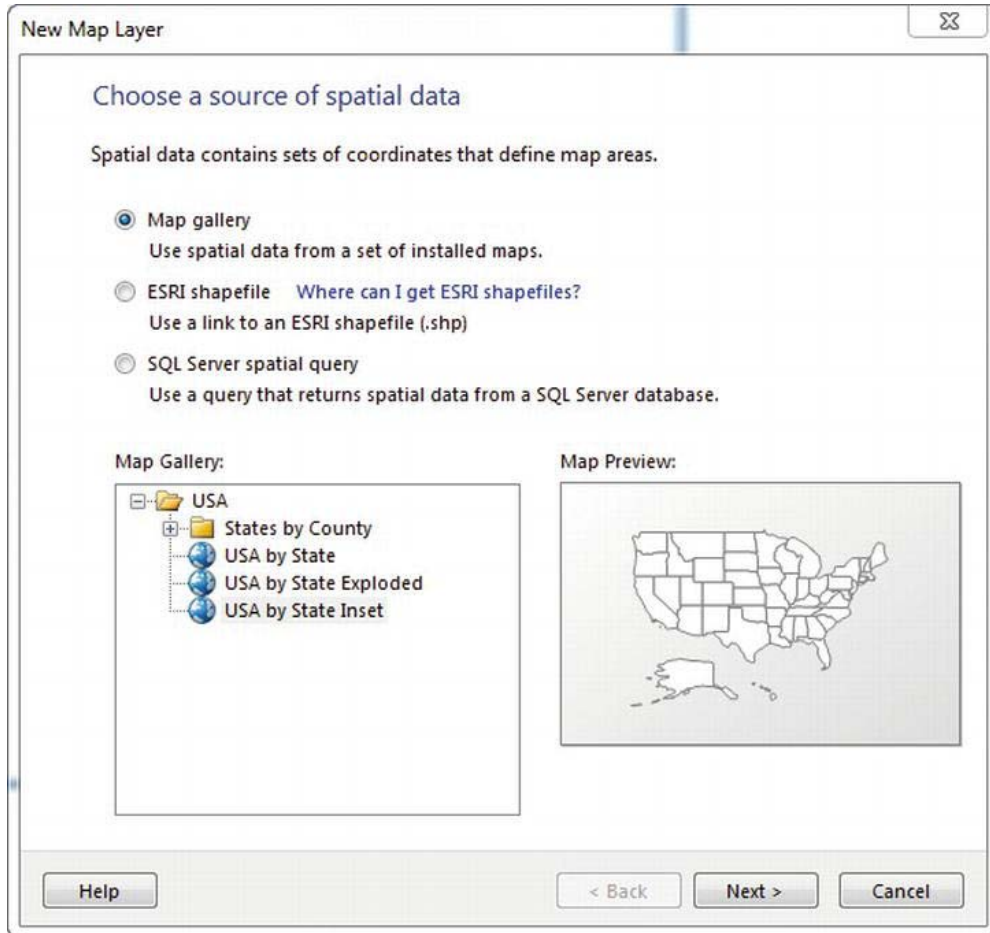


Figure 5-12. *New Map Layer Wizard: Choose source of spatial data*

Click on the Next button to go to proceed to *Choose spatial data and map view options* step of the wizard. On this screen, you have the option to set your zoom level, modify the resolution settings, and add in a Bing Maps layer. For now, let's just keep the defaults on this and click *Next* to continue on to *Choose map visualization* as shown in Figure 5-13.

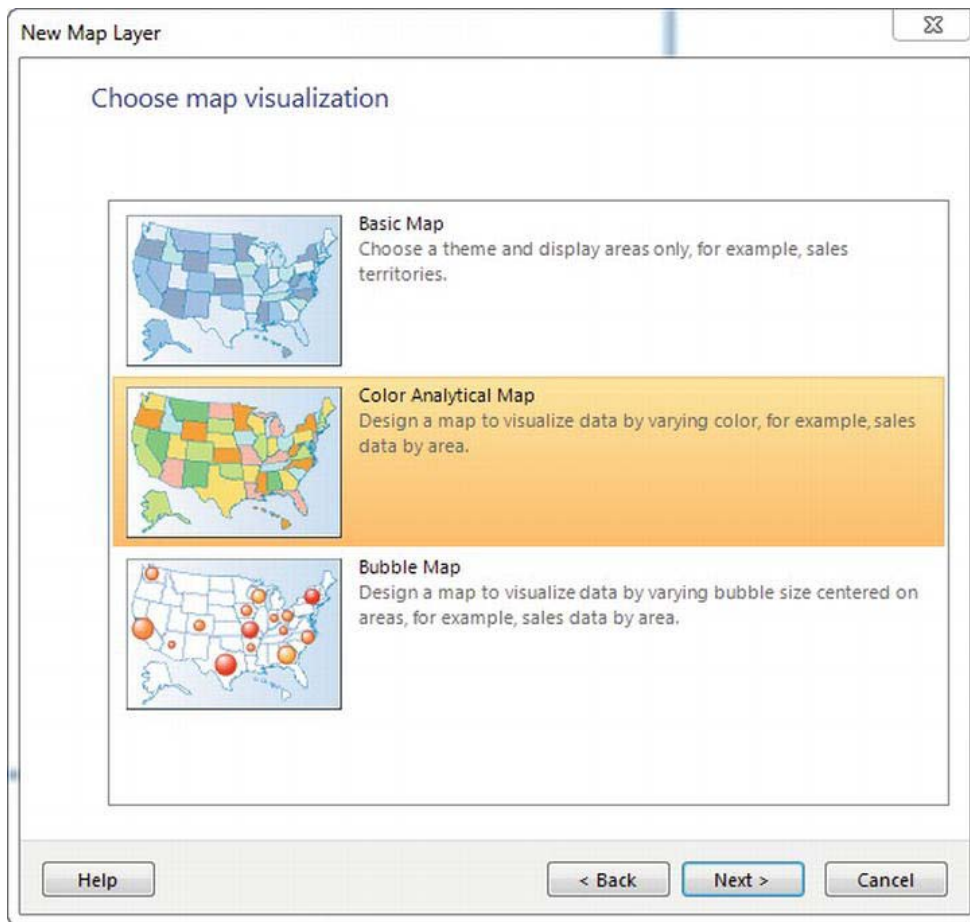


Figure 5-13. *New Map Layer Wizard: Choose map visualization options*

Select the middle visualization Color Analytical Map and click Next. On the *Choose the analytical dataset* screen, you need to select or create a dataset which includes the spatial data. You are going to use the existing dataset, so choose the `Emp_Svc_Cost_By_Patient_State` dataset and click Next to proceed. On the *Specify the match fields for spatial and analytical data* screen, you need to select which fields in your dataset match with Map Gallery that you selected. In this case, the wizard realized that you had a `StateName` field and automatically creates a link to it, as shown in Figure 5-14. If you didn't have `StateName` but you had the state abbreviation, you would need to specify what field in your dataset would match. In that case, you could match your `State` column to the `STUSPS` spatial data column. Be sure that the Match Fields has the checkbox next to `STATENAME` and the Analytical Dataset Fields is set to `StateName` and click Next to proceed to the *Choose color theme and data visualization* screen.

New Map Layer

Specify the match fields for spatial and analytical data

Create a relationship between spatial data and analytical data.

Match Fields	Spatial Dataset Fields	Analytical Dataset Fields
<input type="checkbox"/>	STATEFP	
<input type="checkbox"/>	STUSPS	
<input checked="" type="checkbox"/>	STATENAME	StateName

Spatial data:

STATEFP	STUSPS	STATENAME
25	MA	Massachusetts
31	NE	Nebraska
10	DE	Delaware
11	DC	District of Columbia

Analytical data:

State	Estimated Cost	StateName	Visit_Count	ServiceMo
AK	7850.0000	Alaska	157	ALL Months
AL	12555.2500	Alabama	259	ALL Months
AR	3578.6200	Arkansas	80	ALL Months

Help < Back Next > Cancel

Figure 5-14. *New Map Layer Wizard: Specifying fields to match on*

Modify the Field to Visualize to use [SUM(Visit_Count)], enable the checkbox option to Display labels and set the Data Field to be used as [SUM(Visit_Count)] as well. Figure 5-15 shows the settings on the map's final wizard screen.

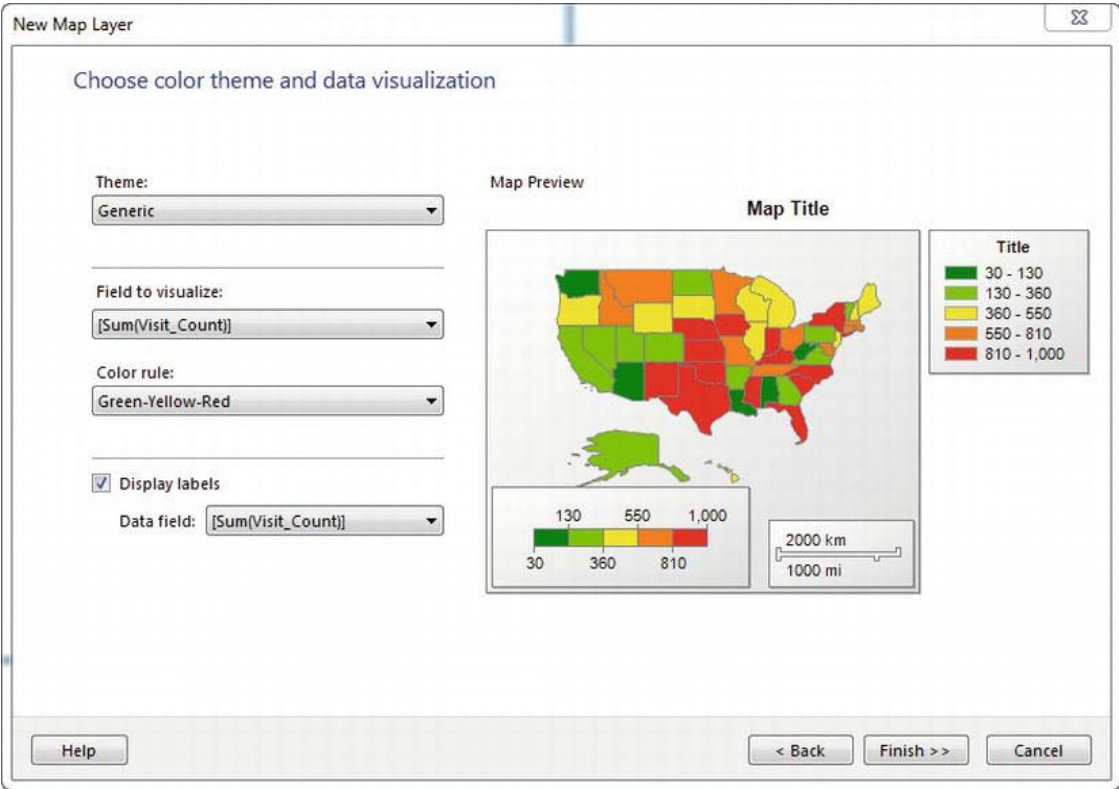


Figure 5-15. *New Map Layer Wizard: Choosing color theme and visualization*

Click Finish to complete the Map wizard and click the Preview tab as shown in Figure 5-16 to see the map in action.

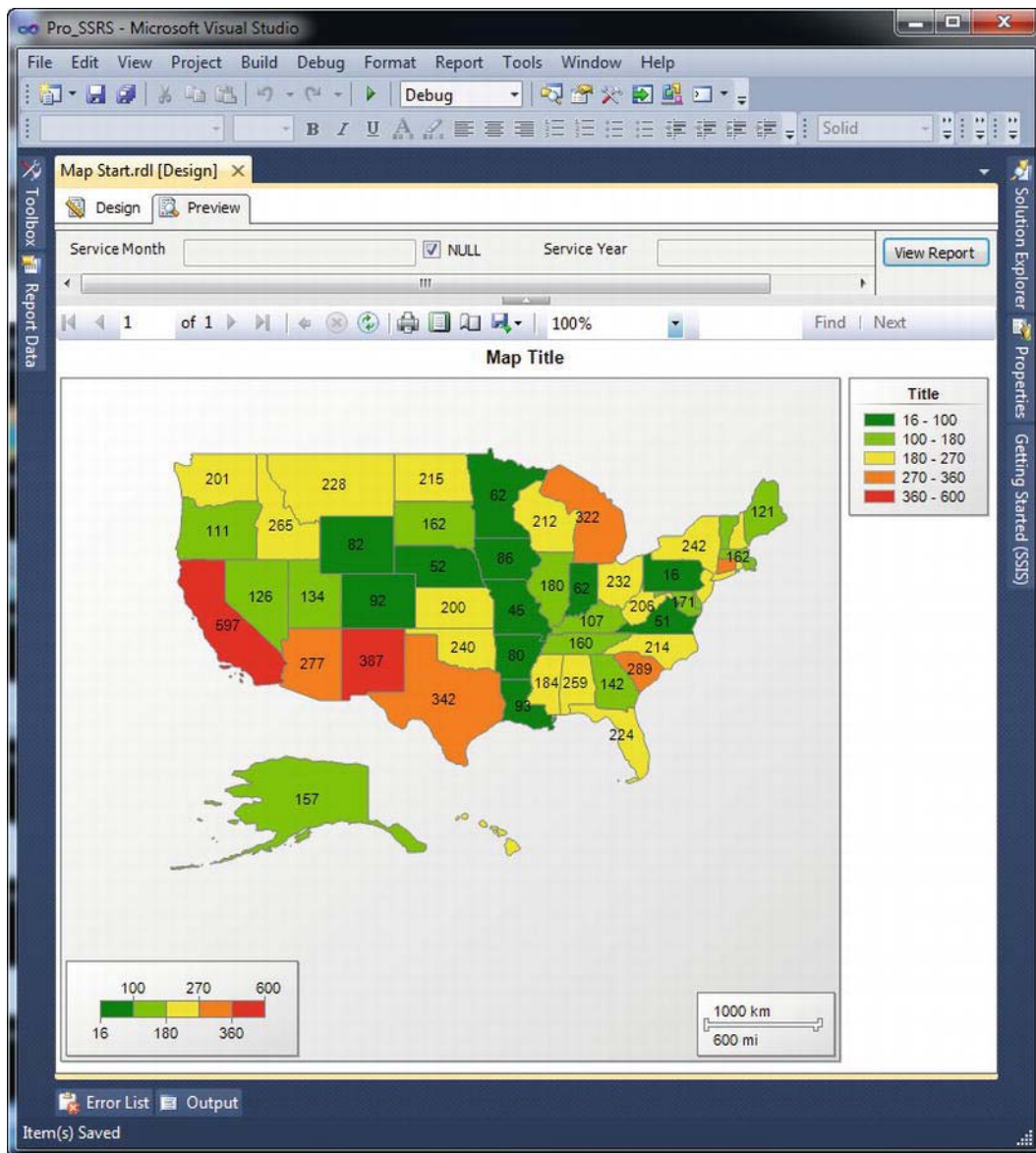


Figure 5-16. Map using spatial query

Listing 5-4 shows sample RDL elements for images.

Listing 5-4. RDL Output for the Map Control

```

<ReportItems>
  <Map Name="Map5">
    <MapViewport>
      <MapCoordinateSystem>Geographic</MapCoordinateSystem>
      <MapProjection>Mercator</MapProjection>
      <ProjectionCenterX>NaN</ProjectionCenterX>
      <ProjectionCenterY>NaN</ProjectionCenterY>
      <MapLimits>
        <MinimumX>NaN</MinimumX>
        <MinimumY>NaN</MinimumY>
        <MaximumX>NaN</MaximumX>
        <MaximumY>NaN</MaximumY>
      </MapLimits>
      <MaximumZoom>4000000</MaximumZoom>
      <MapCustomView>
        <CenterX>46.09375</CenterX>
        <CenterY>63.4690780639648</CenterY>
        <Zoom>100</Zoom>
      </MapCustomView>
      <MapMeridians>
        <Style>

```

The completed Map object report is called Map.rdl and is located in the Pro_SSRS project.

Implementing a Data Bar

Another new addition to Reporting Services 2008 R2 was the Data Bar. For those of you who are savvy Excel users, you may recognize this report object from your tinkering in Excel. This object is essentially a single bar chart (most often displayed horizontally) which allows for a quick at-a-glance, row by row comparison. Longer bars equate to higher values and shorter bars unequivocally mean lower values.

In this example, we are going to start with the report named Data Bar Start.rdl in our Pro_SSRS project. You will notice that there is a dataset named Emp_Svc_Cost_Data_Bar already defined that executes a stored procedure to return the top ten States and the amount of visits for the specified ServiceMonth parameter. We also have a table with only two columns. Expand the Emp_Svc_Cost_Data_Bar dataset and drag the StateName field into the first column of the details row in the table. Next drag the Data Bar report item into the second column of our details row. Upon releasing the Data Bar, you will be prompted to select the type of Data Bar that you want. Your options are:

- Bar
- Stacked Bar
- 100% Stacked Bar
- Column
- Stacked Column
- 100% Stacked Column

For this example, choose Bar as shown in Figure 5-17 and click OK.

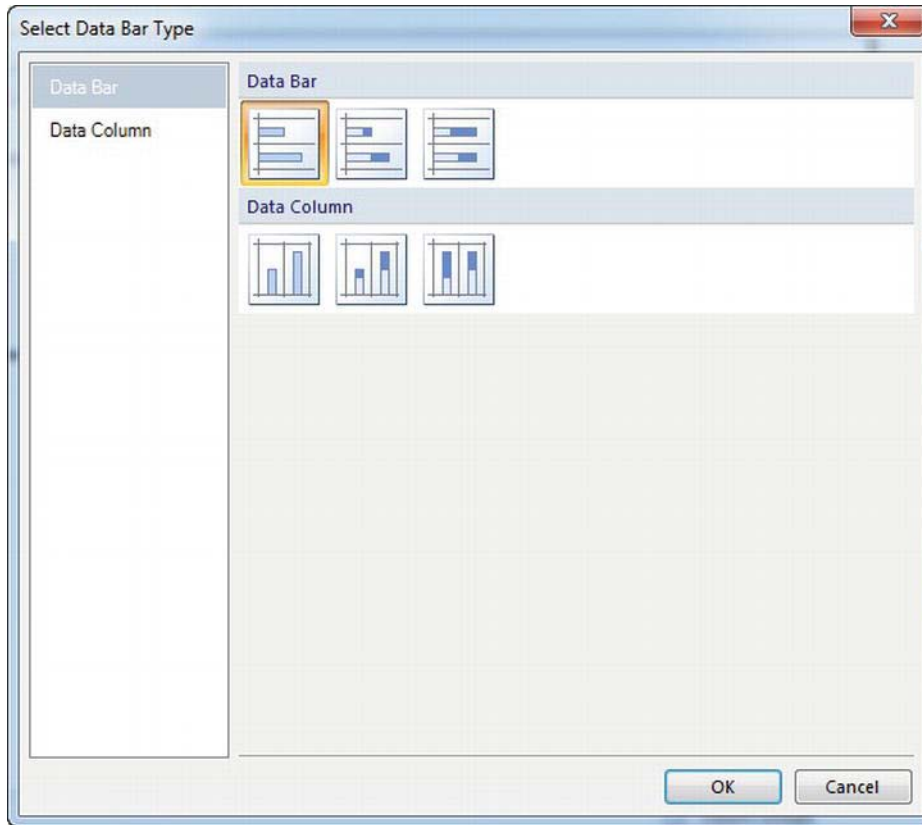


Figure 5-17. Data Bar Type Selection

Next, drag the `Visit_Count_By_State` field over top of the Data Bar and hover over it until the Chart Data dialog box pops up. When you see the Chart Data dialog, drag the selected field into the white space of the Values section, as shown in Figure 5-18.

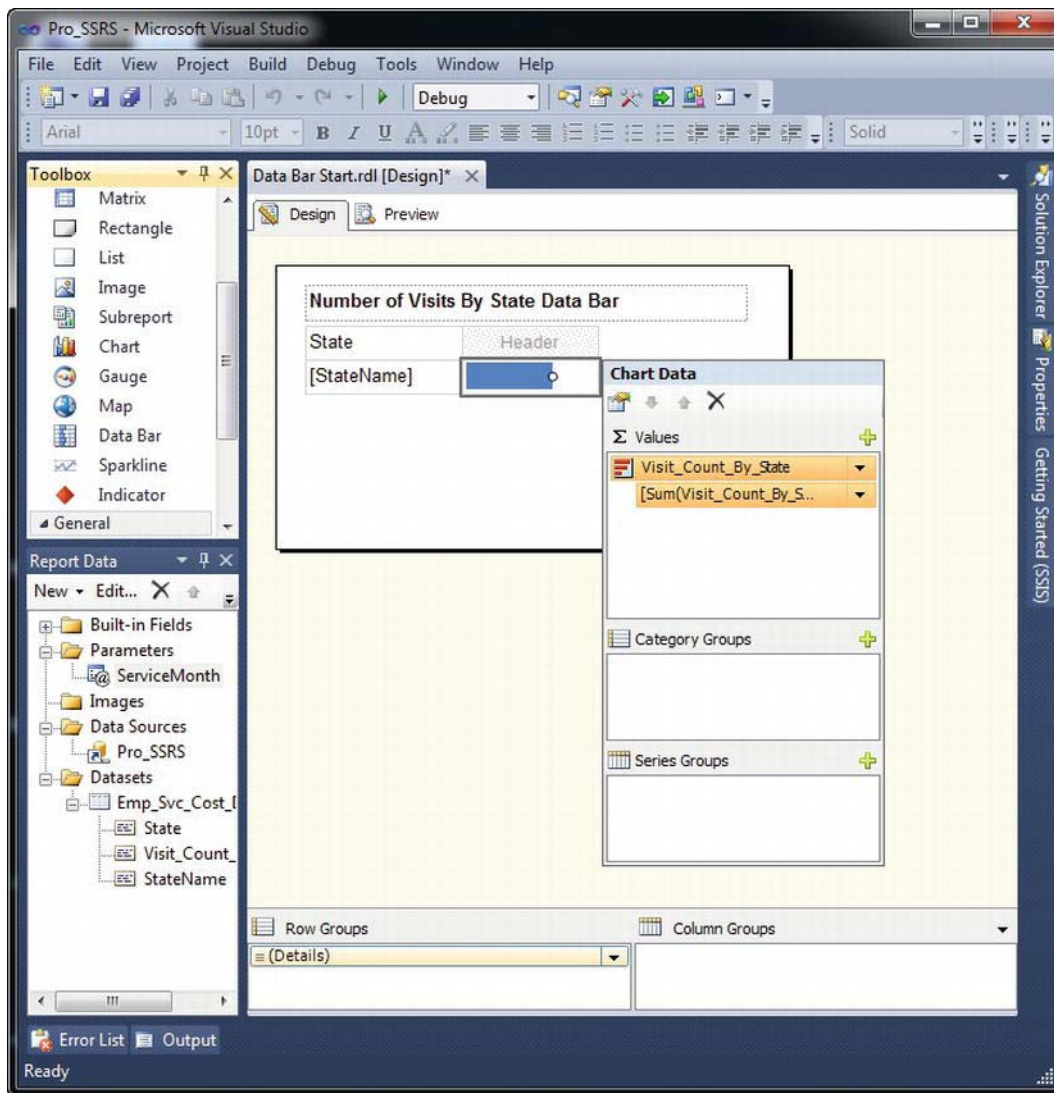


Figure 5-18. Data Bar: Chart Data Value

You could stop here, but let's add a little more pizzazz to the report. Label the header column with the Data Bar "Visit Count." Then make the header row Bold and Centered. Now, select the Data Bar—the blue bar and not the cell—and then right click it to bring up the dialog box and choose Show Data Labels. This will show the values when you run the report. Next, right click the Data Bar again and then select *Series Properties...* Go to the Fill tab, select Gradient with a Gradient style of Left right, as shown in Figure 5-19. Then click on the f_x button next to the primary color (top color) to create an expression.

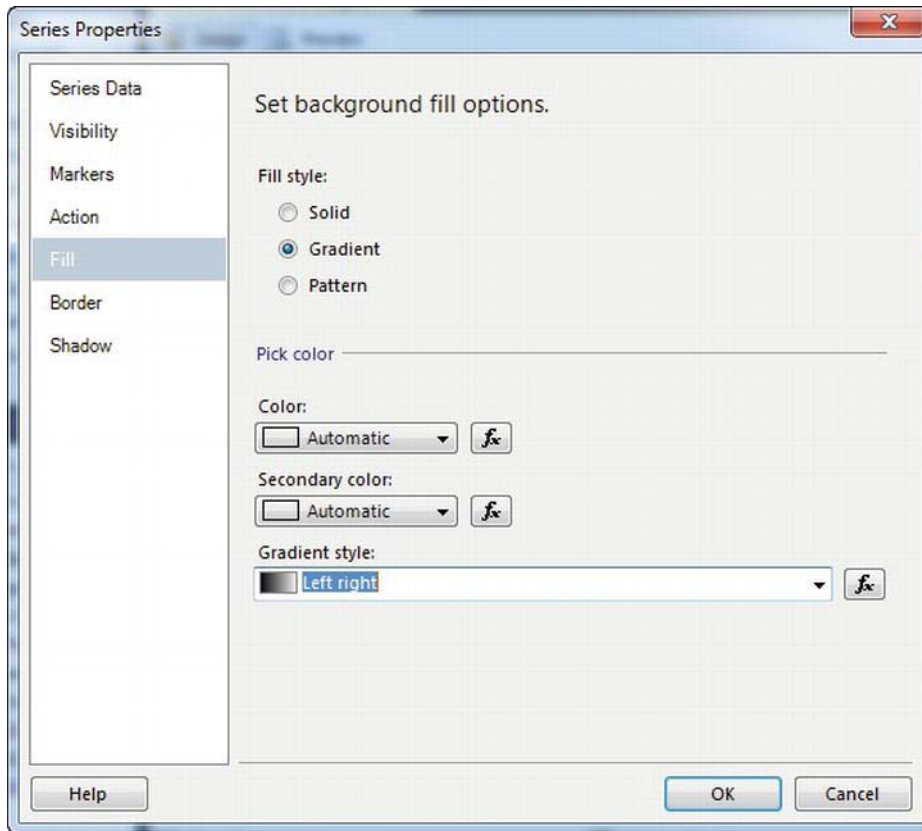


Figure 5-19. Data Bar: Series Properties fill options

Once you have the expression editor open, enter the following expression and click OK:

```
=IIF(Parameters!ServiceMonth.Label <> "ALL",IIF(Fields!Visit_Count_By_State.Value > 75,
"Green", "Red"),IIF(Fields!Visit_Count_By_State.Value > 300,"Green", "Red"))
```

As shown in this expression, you can nest your conditional statements. Here we are checking to see if the parameter label selected was not ALL. Then it checks for the Visit_Count_By_State to see if it is more than 75 for the month selected. If it is, then make the Data Bar green and if not, make it red. The outer IIF statement checks to see if the count is greater than 300 and sets the colors accordingly. Click *Preview* to see your report in action, as shown in Figure 5-20.

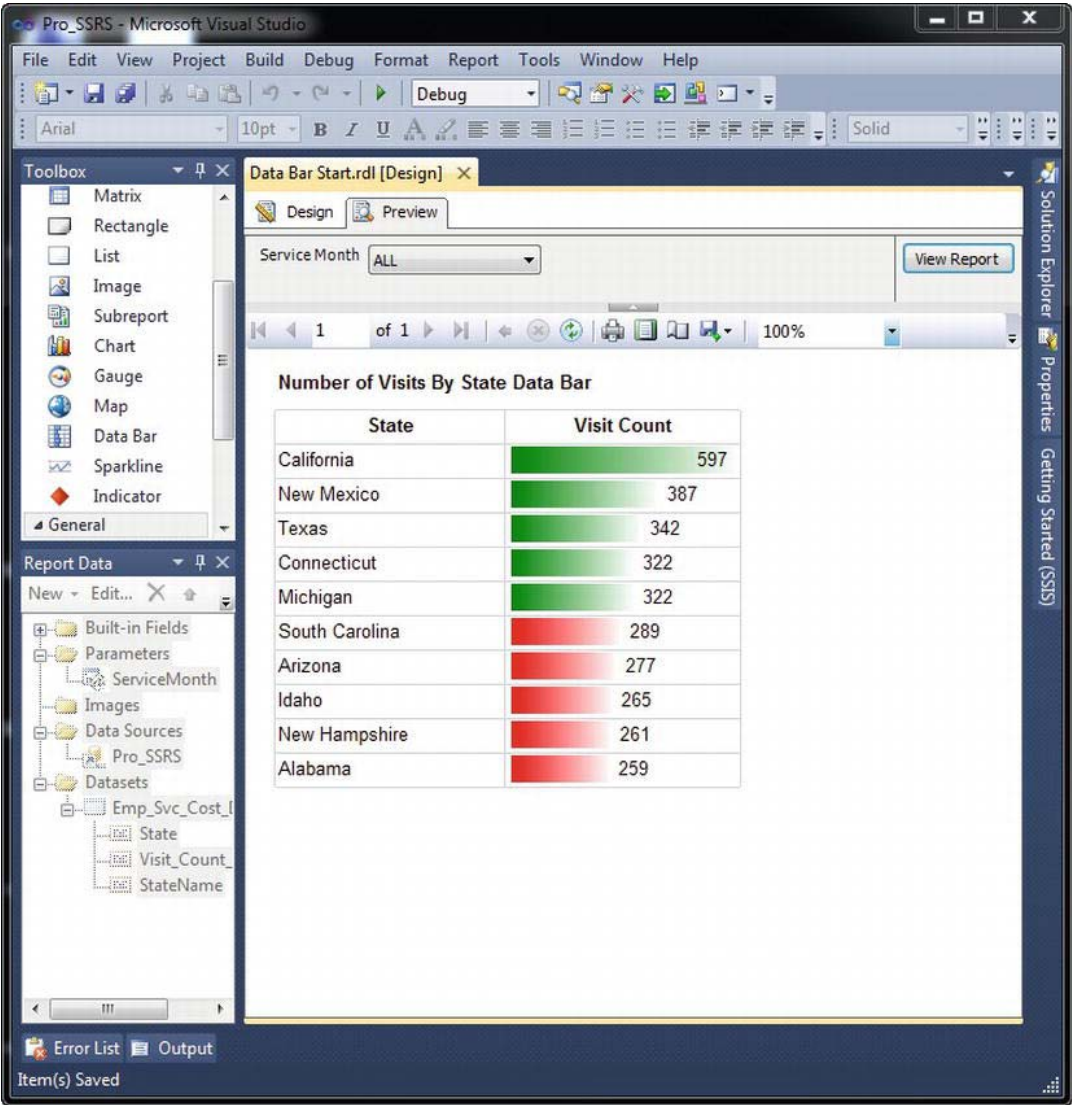


Figure 5-20. Preview of the Data Bar Report

Listing 5-5 shows a section of the RDL file for the Data Bar report item.

Listing 5-5. RDL Data Bar

```

<Chart Name="DataBar1">
  <ChartCategoryHierarchy>
    <ChartMembers>
      <ChartMember
        ...
        <ChartMember>
          <Label>Visit Count By State</Label>
        </ChartMember>
      </ChartMembers>
    </ChartSeriesHierarchy>
    <ChartData>
      <ChartSeriesCollection>
        <ChartSeries Name="Visit_Count_By_State">
          <ChartDataPoints>
            <ChartDataPoint>
              <ChartDataPointValues>
                <Y>Sum(Fields!Visit_Count_By_State.Value)</Y>
              </ChartDataPointValues>
              <ChartDataLabel>
                <Style />
                <UseValueAsLabel>true</UseValueAsLabel>
                <Visible>true</Visible>
              </ChartDataLabel>
              <Style>
                <Color>=IIF(Parameters!ServiceMonth.Label <&gt; "
ALL",IIF(Fields!Visit_Count_By_State.Value > 75, "Green", "
Red"),IIF(Fields!Visit_Count_By_State.Value > 300, "Green", "Red"))</Color>
                <BackgroundGradientType>LeftRight</BackgroundGradientType>
              </Style>
            </ChartDataPoint>
          </ChartDataPoints>
        </ChartSeries>
      </ChartSeriesCollection>
    </ChartData>
  </ChartCategoryHierarchy>
</Chart>

```

The completed report for the Data Bar is called Data Bar.rdl in the Pro_SSRS solution.

Implementing a Sparkline

At least in my experiences, a common report request by leadership has been to show a month over month or year over year trend analysis. One could do this in prior versions of Reporting Services, but it took a little more effort to get the look and feel of a true trending report. Included with 2008 R2 release of Reporting Services was the Sparkline control. This out-of-the-box report item is simple to use and can display vast amounts of data in a small area.

In this example, we will start by opening the report named Sparkline Start.rdl in our Pro_SSRS Reporting Services solution. You will notice that we have created a dataset which executes a stored procedure named Emp_Svc_Cost_Sparkline. This stored procedure returns the Date, Month and Year a particular visit occurred. You'll also find a Tablix containing the month field from our dataset and some appropriate headings.

To start this example off, drag the Sparkline control from the Toolbox and place it in the blank textbox below the Daily Trend header and choose the Area sparkline type. This is the first type under the Area section. The other three types could be Smooth Area, Stacked Area and 100% Stacked Area. Next,

expand the Emp_Svc_Cost_Sparkline dataset and drag the Visit_Count field on top of the new Sparkline. If you hover over the Sparkline before releasing, you will see the Chart Data screen as you have seen in earlier examples. Release the Visit_Count field into the Values section, as shown in Figure 5-21.

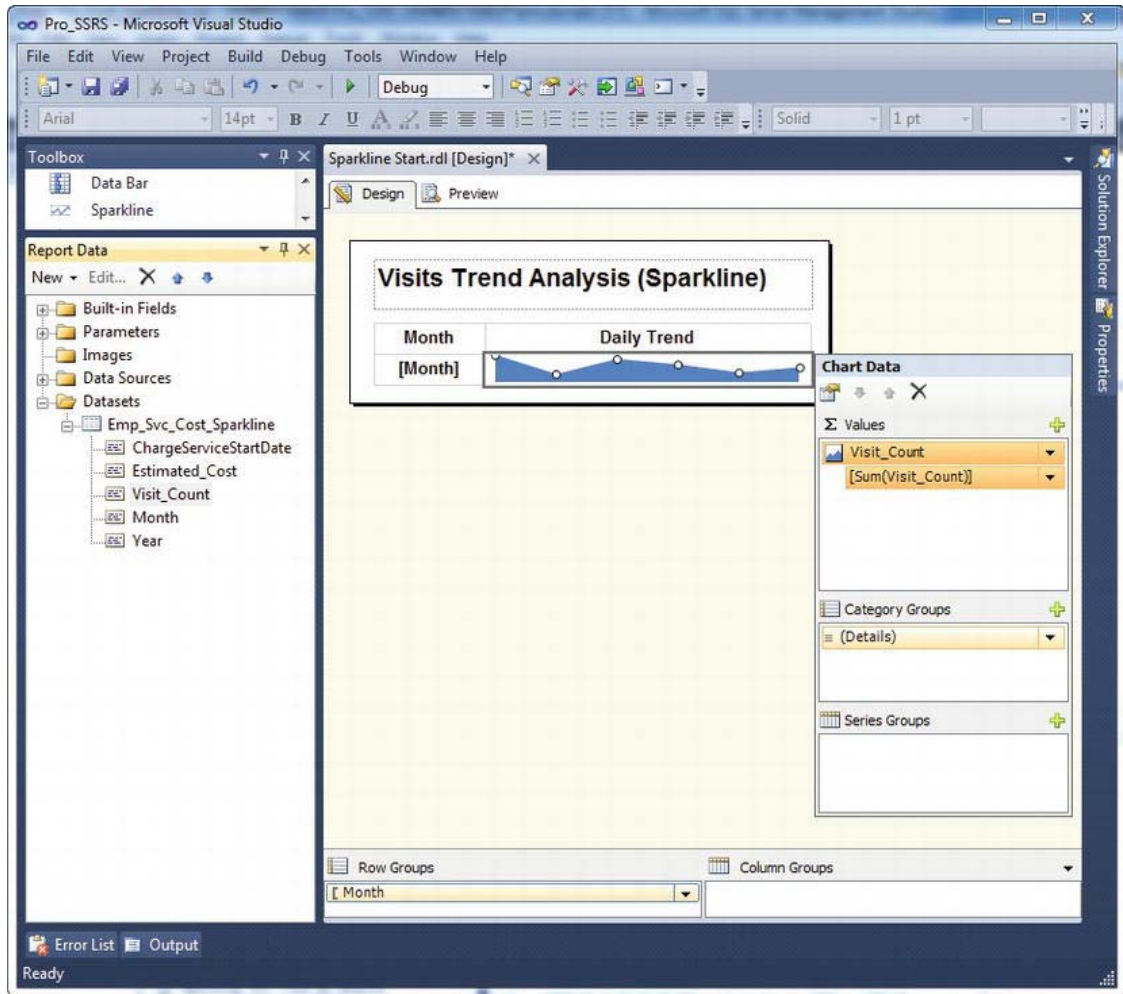


Figure 5-21. Sparkline: Chart data value

Upon previewing the report, you should see a daily trend of visits. It may seem a bit rough right now, but we'll make it a little nicer in a minute. You'll also notice a parameter for MonthlyGoal that we are about to implement as a variation of a key performance indicator or KPI. As we did in the Data Bar example, we are going to change the color of our Sparkline to be a shade of Red or Green, depending on whether we met our monthly goal of visits. Click the Design tab to make some more edits to our Sparkline report.

Double click on your Sparkline control and then right click Visit_Count in the Chart Data window. Choose Series Properties and navigate to the Fill tab. Select Gradient as the Fill Style, White as the Secondary Color and a Gradient Style of Top Bottom. Finally, let's set an expression to change color based on our MonthlyGoal parameter. Click the f_x button to the right of the top color choice to create an expression. Clear out the default text, enter the following expression, and click OK to close the expression editor. The Series Properties window should appear as Figure 5-22.

```
=IIF(SUM(Fields!Visit_Count!Value,"Month") >= Parameters!MonthlyGoal.Value, "Green", "Red")
```

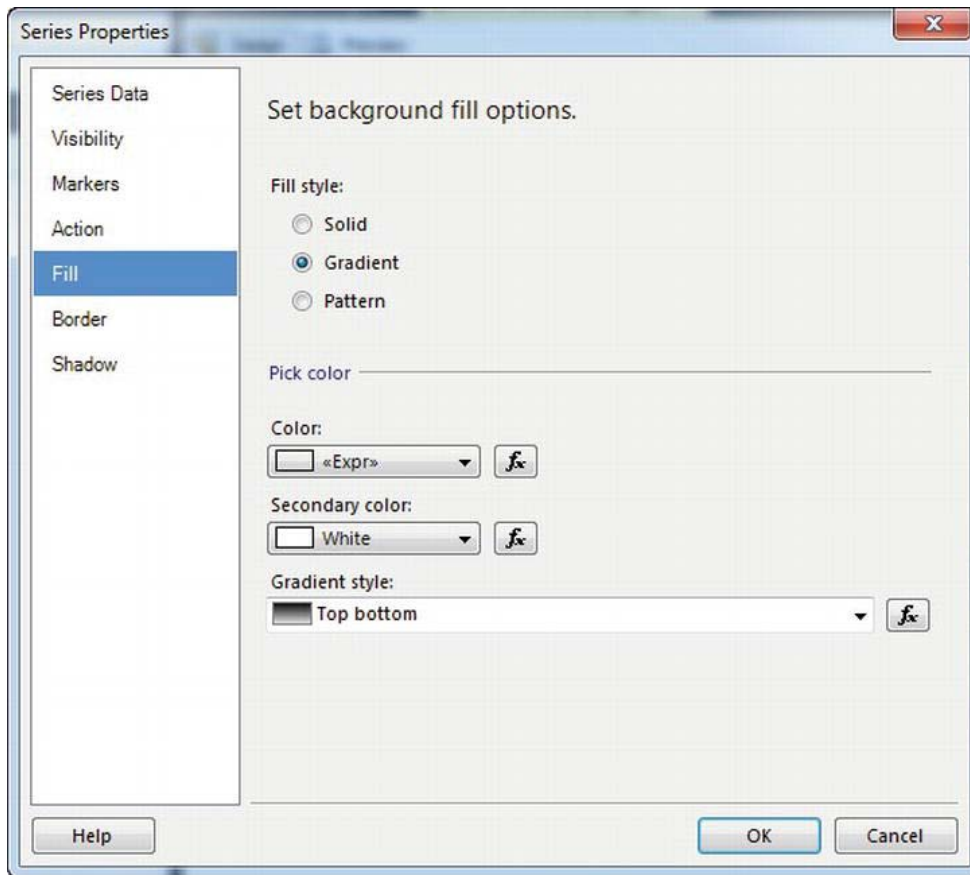


Figure 5-22. Sparkline: Series fill options

We'll be covering expressions later, but this expression is essentially comparing the total count of visits for the month group against the value selected in the monthly goal parameter. Figure 5-23 displays the report in preview mode. Change your Monthly Goal to 2000 and click the View Report button to see the dynamic nature of our color expression. Normally, KPI's are not implemented in this manner, but one could see how to use a value to dynamically change the colors to provide an at-a-glance visual indicator as to the success or failure of meeting predefined goals. Normally, these values would be stored in a database to allow the value to be configurable.

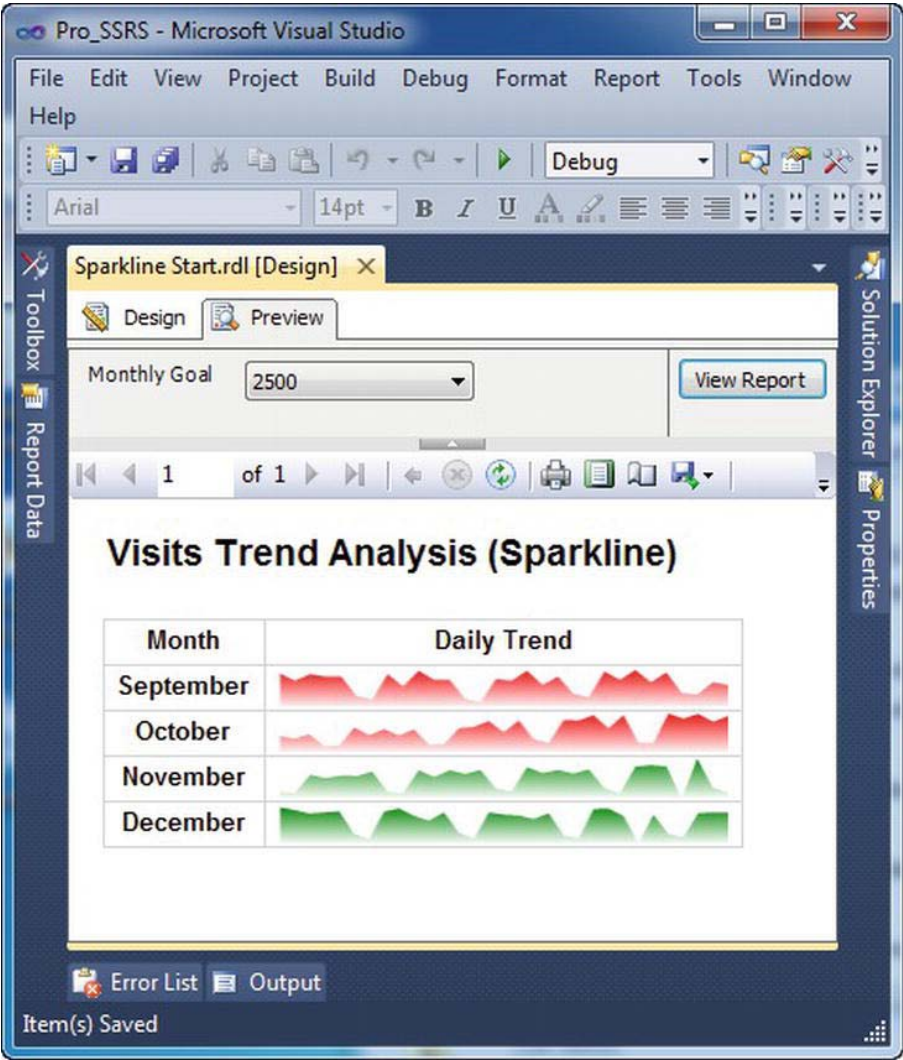


Figure 5-23. Sparkline: Previewing the report

■ **Note** We will go into more detail on this throughout the book, but one thing to take note of, in this example, is that I created a group for Month and removed the Details group. This allowed me to create the trend on a month-by-month basis when the stored procedure returned a day-by-day result set.

Listing 5-6 shows a section of the RDL file for the Sparkline control.

Listing 5-6. *RDL Sparkline*

```
<Chart Name="Sparkline4">
<ChartCategoryHierarchy>
<ChartMembers>
  <ChartMember>
    <Group Name="ChartGroup" />
    <Label />
  </ChartMember>
</ChartMembers>
</ChartCategoryHierarchy>
<ChartSeriesHierarchy>
<ChartMembers>
  <ChartMember>
    <Label>Visit Count</Label>
  </ChartMember>
</ChartMembers>
</ChartSeriesHierarchy>
<ChartData>
<ChartSeriesCollection>
  <ChartSeries Name="Visit_Count">
    <ChartDataPoints>
      <ChartDataPoint>
        <ChartDataPointValues>
          <Y>=Sum(Fields!Visit_Count.Value)</Y>
        </ChartDataPointValues>
        <ChartDataLabel>
          <Style />
        </ChartDataLabel>
        <Style>
          <Color>=IIF(SUM(Fields!Visit_Count.Value,"Month") >=
Parameters!MonthlyGoal.Value, "Green", "Red")</Color>
          <BackgroundGradientType>TopBottom</BackgroundGradientType>
          <BackgroundGradientEndColor>White</BackgroundGradientEndColor>
        </Style>
      </ChartDataPoint>
    </ChartDataPoints>
  </ChartSeries>
</ChartSeriesCollection>
</ChartData>
</Chart>
```

The completed report for the Sparkline is called Sparkline.rdl in the Pro_SSRS solution.

Implementing an Indicator

In Reporting Services 2005 and 2008, we could create the functionality that we see in reports containing KPI's. However, it took a little creativity in the use of expressions. For example, if you wanted to show an image based on a certain value in a result set, you would need to set the visibility of each of the images to be based on an expression. That approach is not too difficult to implement, but it is a more difficult to maintain. In 2008 R2, we were provided this functionality right out of the box with the Indicator report item.

Often times, the consumers of our reports like to see a visual indicator as to the trend in a basic directional fashion. For example, one may want to see if the total visits were higher than last month, the

same or was it less. If we see a consistent trend of negative results, decision makers may want to dig deeper into the possible causes. This is the style of the report that we are going to create in this example. Our decision makers want to see the current state of total visits in relation to the prior month.

In this example, we will start off with Indicator Start.rdl in our Pro_SSRS solution. Included in the Indicator Start report is a dataset that returns four years of sample visit data. You will also notice that there is a tablix with three columns. Two of these already have some of the data needed on the report and a third will be used for our Indicator report item. We have shown you some of the ways to add items to your report design surface, but as with many applications, one can often complete the same task several different ways.

Right click on the empty textbox in Tablix1, select *Insert* at the bottom of the sub-context menu, and choose *Indicator*. When you add an Indicator report item to a report, you are prompted with the Indicator Wizard as shown in Figure 5-24. There are four categories of indicators which you can choose. However, each of these indicators can be customized down to the color and the values used to control them. You can even add more indicator states, use your very own custom images, or embed images within the Gauge control.

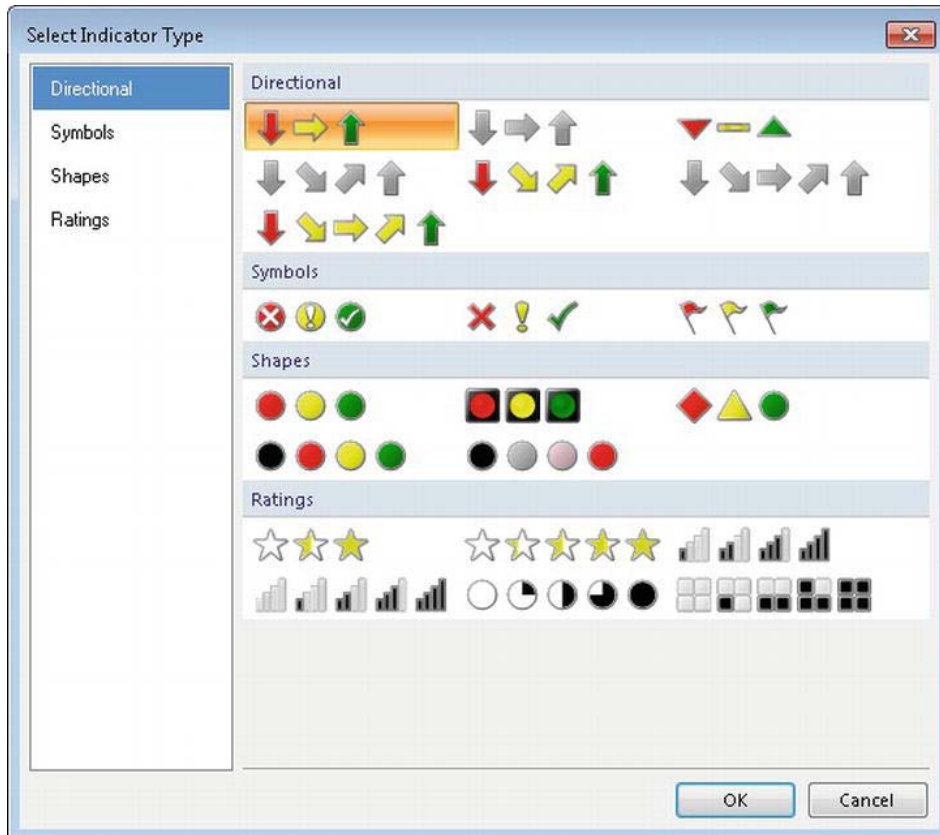


Figure 5-24. Indicator: Indicator type selection

We are going to use the default Directional indicator, the one with three arrows. Be sure that indicator is selected. Then click OK to return to the report designer. At this point, we have added an indicator, but we have yet to tell it what field to use. Right click on the indicator, select *Indicator Properties* and choose the *Values and States* tab. This is the screen where most of the magic happens. Set the Value to be `=SUM(Fields!Diff.Value)`, as you want to display the difference between the current month and prior month as shown in Figure 5-25. Next, switch the *States Measurement Unit* to Numeric from Percentage. If your results were coming back as a percentage of change value, then you would leave this at its default setting. If you wanted to add or reduce the number of indicator states, you could do that by clicking the Add or Delete buttons, respectively. Three values will suit our needs in this example, so make the following changes to the start and end values:

- Red Down Arrow – Clear the start value and set -1 as the end. This essentially is telling SSRS that anything negative should be represented by the red down arrow.
- Yellow Right Arrow – Set the start and the end values to 0 to represent no change in the current and prior total visit counts.
- Green Up Arrow – Enter a 1 in the start value and clear out the end value to show all positive values as an increase in visits over prior month.

Indicator properties

General

Value and States

Action

Change indicator value

Value: [Sum(Diff)]

States Measurement Unit: Numeric

Indicator states:

Add Delete

Icon	Color	Start	End
Red Down Arrow	Red		-1
Yellow Right Arrow	Yellow	0	0
Green Up Arrow	Green	1	

Help OK Cancel

Figure 5-25. Indicator properties

After you have made the required changes, click OK to return to the Indicator Start report. Click the Preview tab to run the report. If all settings are set appropriately, your report should look like Figure 5-26.

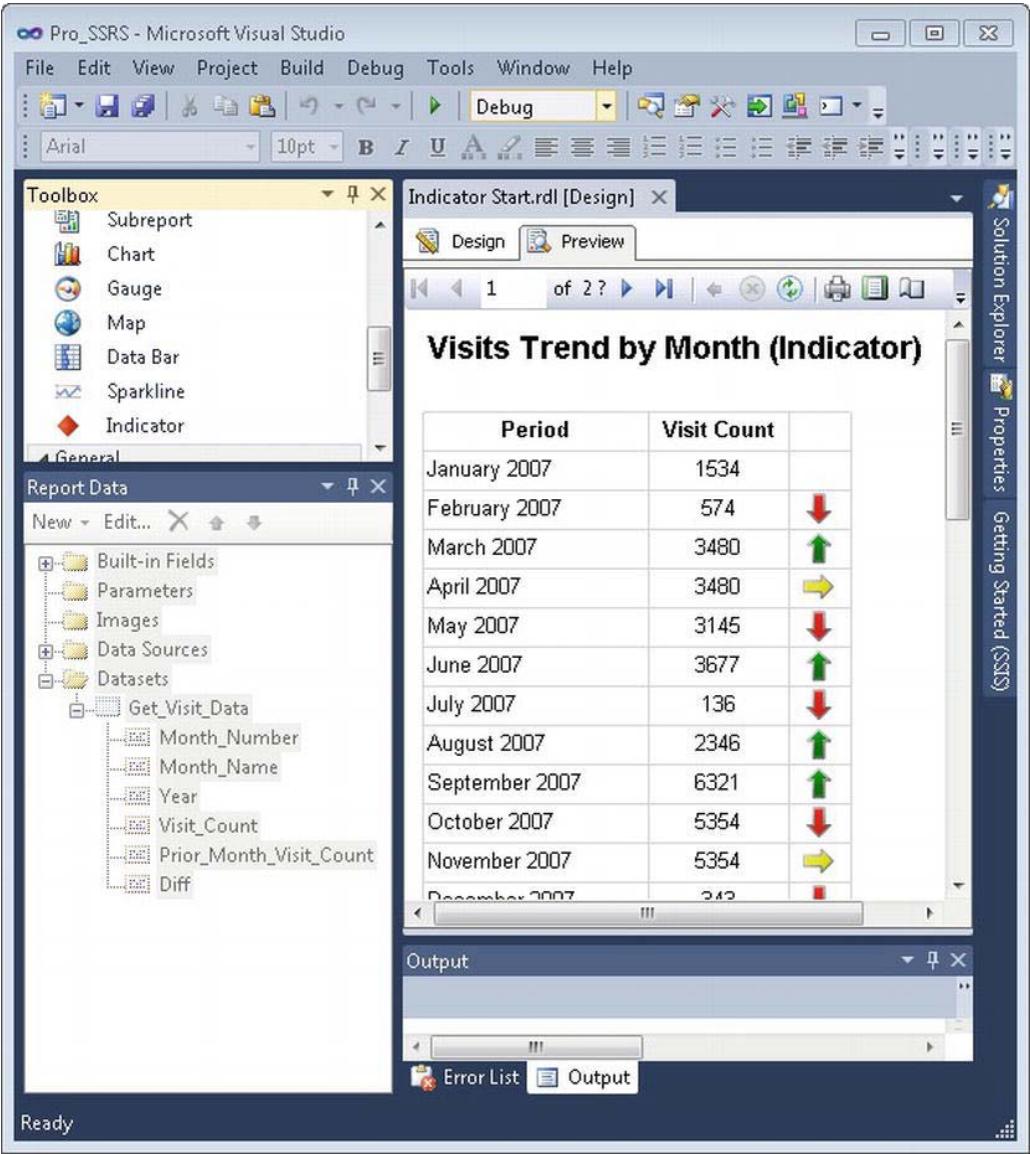


Figure 5-26. Preview Indicator Start

Listing 5-7 shows a consolidated section of the RDL file for the Indicator report item.

Listing 5-7. RDL Indicator

```
<StateIndicator Name="Indicator1">
  <GaugeInputValue>
    <Value>=Sum(Fields!Diff.Value)</Value>
    <Multiplier>1</Multiplier>
    <DataElementOutput>NoOutput</DataElementOutput>
  </GaugeInputValue>
  <TransformationType>None</TransformationType>
  <TransformationScope>Tablix1</TransformationScope>
  ...
<IndicatorStates>
  <IndicatorState Name="ArrowDown">
    <StartValue>
      <Value />
      <Multiplier>1</Multiplier>
    </StartValue>
    <EndValue>
      <Value>-1</Value>
      <Multiplier>1</Multiplier>
    </EndValue>
    <Color>Red</Color>
    <ScaleFactor>0.99</ScaleFactor>
    <IndicatorStyle>ArrowDown</IndicatorStyle>
    <IndicatorImage>
      <Source>External</Source>
      <Value />
    </IndicatorImage>
  </IndicatorState>
  <IndicatorState Name="ArrowSide">
    ...
  <IndicatorState Name="ArrowUp">
    ...
  </IndicatorState>
</IndicatorStates>
```

The completed report for the Indicator is called Indicator.rdl in the Pro_SSRS solution.

Summary

In this chapter, we covered some of the more visually appealing objects in Reporting Services, such as charting, maps, and gauges. Most of these objects are typically found in dashboard style reporting to provide an at-a-glance perspective. We also went over the chart data region and walked through examples of the different chart types available. Now that you're more comfortable with the design environment, you'll learn how to use it to design and deploy some real reports. In the next chapter, we'll show how to take a step-by-step approach to adding these report items to a report that was designed as part of an SSRS migration for a health care application.

Building Reports

In the previous chapter, you laid the foundation for your first report by creating a query and subsequent stored procedure. You also learned about the fundamental elements used to build reports and are now familiar with the design environment. Now, it is time to put all the pieces together and begin building reports. You can easily apply the concepts introduced in this chapter to any company that uses SQL Server and relational database systems. This chapter will focus primarily on creating a reporting solution based on data from a SQL Server health care database; it will use many of the report elements that have been available since SSRS's inception in SQL Server 2000 through 2012. SSRS 2008 and 2008 R2 introduced many new features such as report variables, enhanced charting, and data visualizations. The most significant additions were the Tablix data regions and dashboard style elements like Sparklines, Data Bars, and the Indicator. We are excited to incorporate these significant and long-awaited enhancements into the reports featured in this book.

The report you will be creating in this chapter is called the Employee Service Cost report. This report will utilize the same query and stored procedure, `Emp_Svc_Cost`, which you have been working with since Chapter 2 to provide the report data. As a reminder, the query returns detail records that represent services performed for patients, such as visits by skilled nurses or home health aides. Each type of service has an associated cost for the health care company. This report, when complete, will show important cost points based on associated data provided by the query, such as the patient's diagnosis, the employees who performed the services, the date of each service, and the branch location of the patient. By grouping and sorting the report at these cost points, you will be able to see the cost of services from the individual patient all the way up to the branch location, which might serve hundreds of patients. You will group and calculate the cost amount at each level.

Specifically, in the following sections, you will create the Employee Service Cost report initially with the Report Wizard, which produces a report based on predefined selections, and then from scratch. We will show the process of using the wizard for demonstration purposes only and therefore will not continue with the report that it produces. For the report that you build from scratch, you will add all the features that the Report Wizard can add, plus much more. The following list highlights the design goals for the Employee Service Cost report:

- Step through adding a base report that uses the Table data region based on the dataset you defined for the `Emp_Svc_Cost` query.
- Add several basic formatting elements to the report.

- Add interactivity to the report, with document mapping, visibility, hyperlink actions, and interactive sorting (introduced in SSRS 2005). Both document mapping and hyperlink actions allow the user to navigate to defined locations either within the report or outside the report, such as a Website. In this chapter, you will use visibility properties within your report to expand and collapse report items from summary to detail. Interactive sorting gives an SSRS report versatility by allowing it to be sorted in much the same way that Microsoft Outlook allows sorting by clicking column headers.
- Add parameters to the report automatically by changing the dataset from a query to a parameterized stored procedure. You will also add other datasets to populate the parameters defined by the stored procedure.
- Learn how to use multivalued parameters using a modified stored procedure and UDF.
- Add a filter to the Table data region to show only service types that are visits.
- Add a Chart data region for the top ten diagnoses to the report.
- Add a report variable (introduced in 2008) to use as a constant threshold value.
- Add a gauge control to the report to show threshold information.
- Add a column grouping to the report Table region. Column groupings within the Table region are part of the new Tablix functionality for the Table, Matrix, and List data regions. The Tablix functionality was covered in detail in Chapter 4.
- Add the final touches to the report, such as a page header and footer, title, and page numbers.

In addition, as you begin to work more closely with report and query parameters, you will learn how to use multivalued parameters. As mentioned in previous chapters, multivalued parameters require special consideration when designing the underlying query. Therefore, in this chapter, you will use a modified version of your stored procedure that takes advantage of a UDF; this will teach you how to best utilize this feature.

In the preceding chapters, we covered the steps for creating the solution, project, and data source that your report will use, so we will not cover these steps again here. We will, however, show how to use the same data source properties to connect to the health care database where the data for your report reside. The same database also contains the stored procedure you created in Chapter 2, `Emp_Svc_Cost`, which you will use later in this chapter.

Creating a Report with the Report Wizard

In many scenarios, the Report Wizard is a fast method for creating a basic report that can be further enhanced before deployment. The Report Wizard is suitable for reports that are primarily data listings that do not require much special formatting. In this section, you will step through the Report Wizard to create the Employee Service Cost report before designing the same report manually.

To open the Report Wizard in your report project, right-click the Reports folder in the Solution Explorer and select Add New Report. The first wizard screen defines the data source. For this example, check New Data Source; however, you also have the choice to use a shared data source that has already been defined as part of the project. Supply the same data source information as you did in the previous

chapter to connect to the Pro_SSRS database. The connection string should look similar to the following:

```
Data Source=localhost;Initial Catalog=Pro_SSRS
```

The next screen in the wizard defines the query. Paste the query you created in Chapter 2 into the Query String area (see Figure 6-1). You can open this query from the Query folder in the code download for the book. The file is called Report_Wizard_Query.sql. Clicking the Query Builder button launches the graphical query designer.

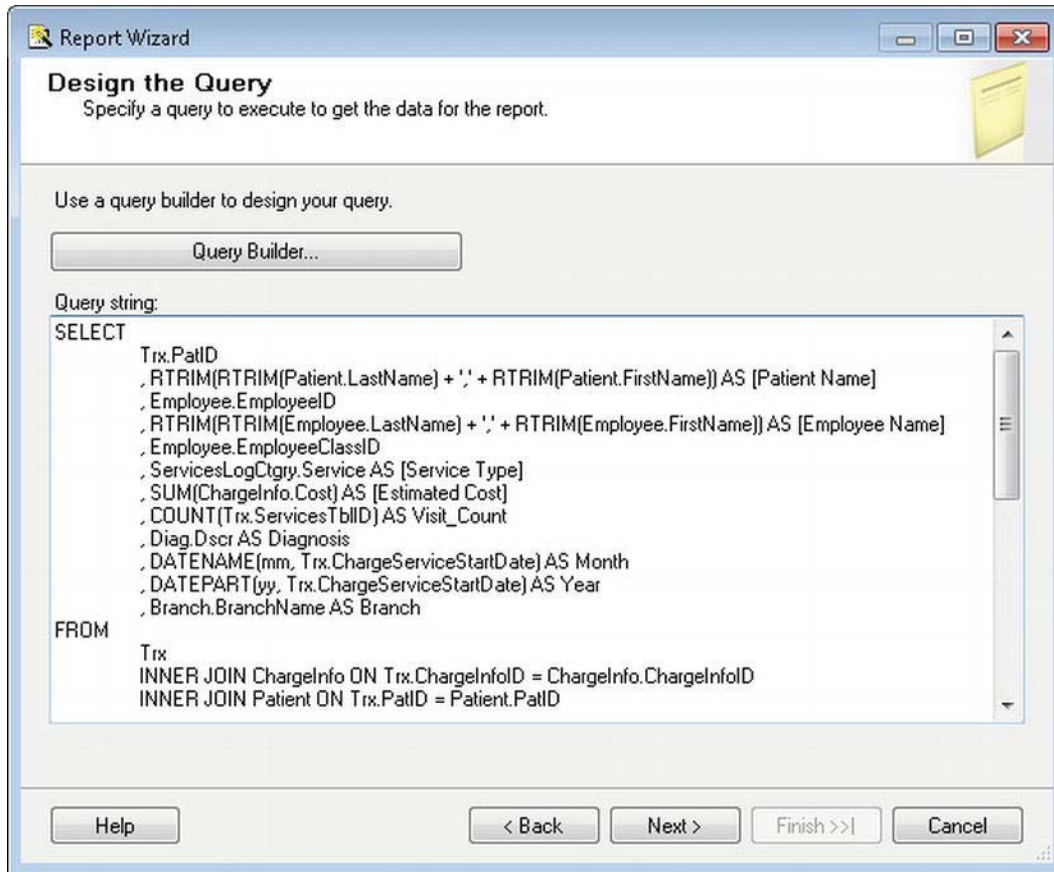


Figure 6-1. Pasting the query in the Query String area

The next screen of the Report Wizard asks whether the report should be in tabular or matrix form. Selecting Tabular will trigger the wizard to provide grouping information on the next screen; selecting Matrix will provide a similar screen for rows and columns instead of groups. For this example, select Tabular, click Next, and choose the grouping and detail layout to show Year as the primary group, with Month, EmployeeClassID, and Employee_Name next. For details, you want to see the patient-specific information—Diagnosis, Visit_Count, Estimated_Cost, and Service_Type—as shown in Figure 6-2.

Once you have grouped the data that will be in the report, the next two screens are primarily for formatting. Here, you can specify whether you want the report to have a stepped or block layout, as well as whether the report will include subtotals and provide drill-down functionality. You can also choose a custom style for the report.

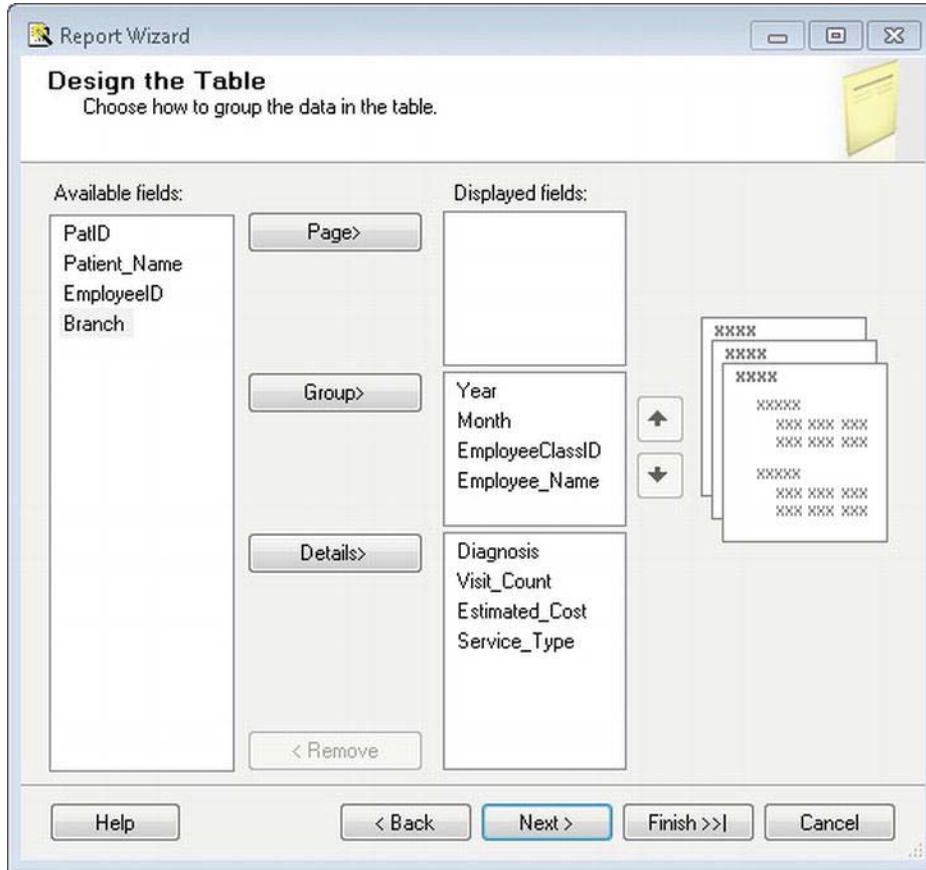


Figure 6-2. Report Wizard group and details selections

For now, choose Stepped with No Drill-Down functionality and click Next. On the Table Style screen, apply the Corporate style to the report, click Next in the wizard, and change the name from the default, ReportX, where X is the next number in sequence of created reports, to **Employee Cost Report Wizard**. Next, check the Preview Report box to have the report executed upon creation, and then click Finish to generate the report. After a few moments, the resultant report appears. Although at first glance it appears to need many cosmetic changes, such as extending the size of the several columns (e.g., Employee Name and Diagnosis), resetting the background color of the groups to White, the font color of [Year] to Black, and formatting the Estimated Cost column for currency, the report is at least functional. Depending on the desired layout, it could need a good deal of work to get it the way you want it. Assuming you accepted one of the default styles, this would be a good starting point for modification, as shown in Figure 6-3.

To take full advantage of the flexibility of SSRS and the report design environment of BIDS, let's create the same report from scratch.

Year	Month	Employee Class ID	Employee Name	Diagnosis	Visit Count	Estimated Cost	Service Type
2009							
	April						
		CNA					
			MCDONALD, KYLIE				
				ALZHEIMERS DISEASE	1	\$50.00	HOME HEALTH AIDE
	December						
			GREEN, JENNY				
				ALZHEIMERS DISEASE	1	\$50.00	HOME HEALTH AIDE

Figure 6-3. Report generated from the Report Wizard

Building Reports from Scratch

When working with a blank report, the first decision will be to choose which data regions to use in the body of the report. This decision is driven mostly by the type of data you are working with and by the report audience. For example, a chief executive officer (CEO) might not be concerned with details, preferring to see summary information about the status of the business products and services, and therefore would be more inclined to view a matrix report with column and row totals. However, in the initial report, you will be working with the Table data region because you want to show the interrelationships between patients and employees with multiple groupings in tabular rows, not columns. The final modification to the report that you will build in this chapter will include column and row groupings to show you how to incorporate elements of the Tablix data region.

In this section, you will follow specific steps to get your report to a basic starting point by adding a Table data region, and then you will continue to add formatting and functionality. When you are finished, the report will contain many SSRS features, including interactive drill-down and navigation links, custom formatting, interactive sorting, populated drop-down parameters, and a chart that displays the top ten diagnoses by cost. You will finish the report by adding several design touches, such

as page numbers and execution times. You will examine how to modify the report to work with multi-value parameters. You will also explore another feature added in SSRS 2008: report variables. For this project, you will add a new report from the Solution Explorer and create a dataset that uses the same query from the previous section of this chapter for the Report Wizard. For simplicity, we have included the starting-point report in the Pro_SSRS project. The EmployeeServiceCost_Start report already has the datasets and initial query defined for the localhost SQL Server, which should match your environment. You will begin by using just the basic query, not the stored procedure. The dataset you will use to begin within the EmployeeServiceCost_Start report is called Emp_Svc_Cost. Later, in the “Setting Report Parameters with Stored Procedures” section, you will modify the dataset to use the stored procedure and see how the parameters defined in the stored procedure will automatically create the report parameters.

In the following sections, you will go through several steps to add functionality to a single report. The steps are provided so that you can walk through the process of building the report, starting with the EmployeeServiceCost_Start report; however, at several intervals, you may choose to open one of the several sample reports that reflect the completed steps. If a report is available, we will point it out in the text.

With the EmployeeServiceCost_Start report open in BIDS, move to the Design tab. The following steps get you to your starting point in the report, where you will begin to apply more advanced formatting and logic:

1. Drag the Table report element to a blank section of the design grid.
2. On the Report Data pane, drag the fields—Estimated_Cost and Visit_Count from the Emp_Svc_Cost dataset—to the detail row in the order listed. Delete the extra column by right clicking on the column and selecting Delete Columns. Notice that the column headings—Estimated Cost and Visit Count—were automatically created for you for each field dragged to the detail row. If you do not see the Report Data Pane, you can pull it up by using CTRL + ALT + D or by selecting it under the View menu.
3. Edit the Visit_Count and Estimated_Cost field expressions to be sums, as in =Sum(Fields!Estimated_Cost.Value) by right clicking on each detail field and choosing Expression in the submenu. You will notice that when you develop your report in BIDS, each value that you enter into the report regions will be enclosed with brackets or be represented as “<<expr>>”. For example, the expression =Sum(Fields!Estimated_Cost.Value) will be visually represented as [Sum(Estimated_Cost)] in the table cell because the expression is a known value, even though it is an expression itself. You can also highlight the field, right click, and then select Sum under the Summarize By... submenu.
4. Drag the Employee_Name field down to the Row Groups pane and release it above the (Details) group. This will create a grouping on Fields!Employee_Name.Value and add a new column to the Table. By default, the group is named the same as the field being grouped on. You will also notice that a vertical dashed line is placed between the details section and the new Employee_Name column of our Tablix: this gives you a visual indicator of where the group section ends and the details section begins.
5. Next, drag the Patient_Name field and release it above the Employee_Name group to create a row group on Patient_Name. Do this step for Service_Type and Diagnosis with each one being above the former. This will create a hierarchy of Diagnosis, Service_Type, Patient_Name, and Employee_Name, respectively.

6. Right click on the (Details) group in the Row Group pane and select Group Properties. Click Add under Group Expressions and select [Employee_Name] in the Group On drop-down list.
7. To give our report a clean stepped look, right click on the Employee_Name field in the table, select Insert Row and then Outside Group – Above – as shown in Figure 6-4. Next, drag the Patient_Name field that was added in step 5 into the new empty cell above Employee_Name. Right click on Patient_Name and then do the same thing by selecting Insert Row and then Outside Group – Above. This time, drag the Service_Type that was added in step 5 into the new empty space. With two of our groupings down, let's do the same thing for the next level up in our hierarchy. Right click on Service_Type and Outside Group – Above. This time, drag our Diagnosis field above the Service_Type. Figure 6-5 shows what our report should look like in the designer.

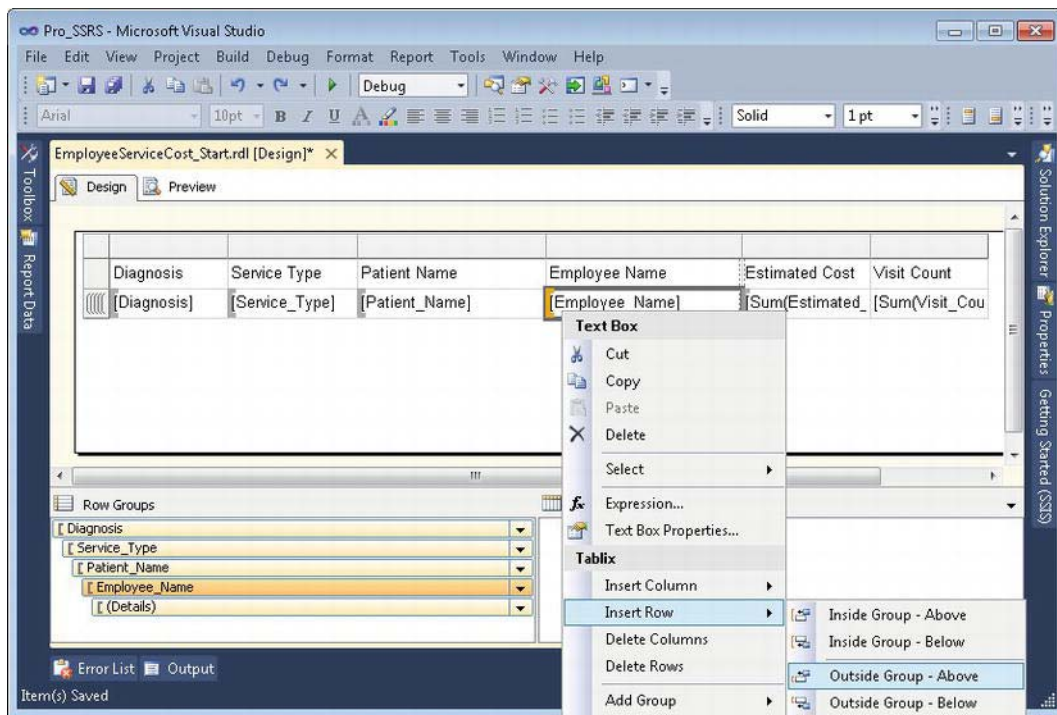


Figure 6-4. Employee Service Cost report Add Outside Group – Above for stepped look

8. Now, remove the three columns with headers that read Diagnosis, Service Type, and Patient Name by right clicking each of them and choosing Delete Columns. After you remove the three columns, drag your Tablix closer to the left side of the Design pane. Then make the column labeled Employee Name about three inches wide so that the values don't end up wrapping to the next line.

9. Select the textbox containing the [Service_Type] in our table and hit F4 to show our Properties Window. Scroll until you see Padding and expand the properties by clicking the arrow. Modify the Left padding to be 10pt. Do the same thing for [Patient_Name] and [Employee_Name] but set them to 20pt and 30pt, respectively.

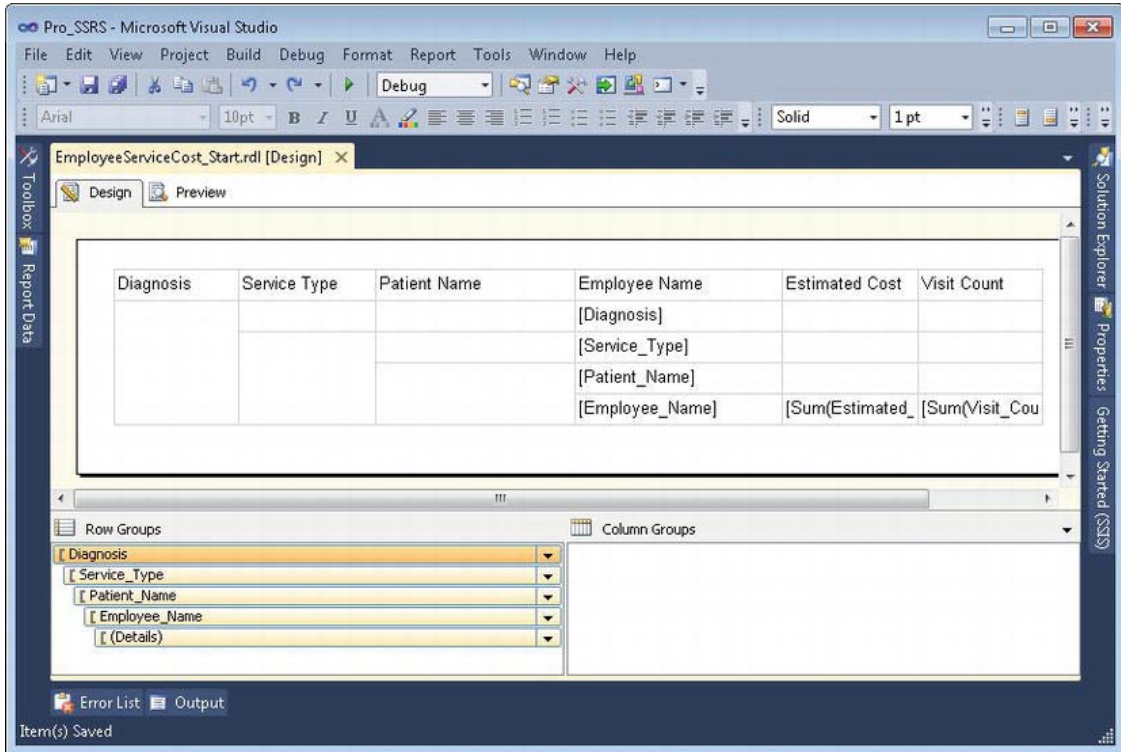
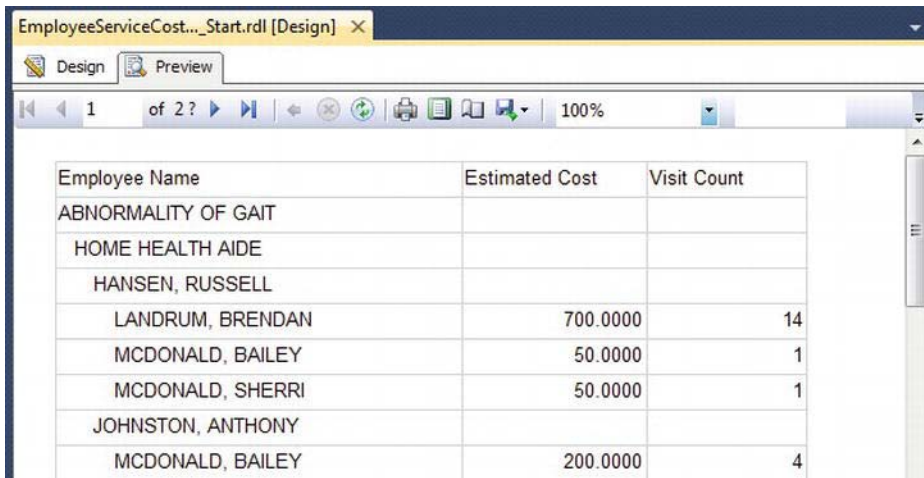


Figure 6-5. *Employee Service Cost report Outside Groups- Above*

After these nine steps, the report is starting to take form, as you can see in the preview. Although not yet aesthetically appealing, it displays the data in the appropriate, hard-fixed groupings and is tabulated so that it is easy to discern the detailed service information, such as the cost and counts of services for each patient (see Figure 6-6).



Employee Name	Estimated Cost	Visit Count
ABNORMALITY OF GAIT		
HOME HEALTH AIDE		
HANSEN, RUSSELL		
LANDRUM, BRENDAN	700.0000	14
MCDONALD, BAILEY	50.0000	1
MCDONALD, SHERRI	50.0000	1
JOHNSTON, ANTHONY		
MCDONALD, BAILEY	200.0000	4

Figure 6-6. Employee Service Cost report details and groups

Formatting the Output

You can modify several quick and easy report properties to add a more professional look and feel to the report:

- Border Style
- Format

By using the Shift or Control key, or by clicking and dragging the mouse, it is easy to apply report properties to many cells simultaneously. For the Estimated Cost and Service Count header cells, you will add a border to the bottom, separating the record header from the actual data. First, highlight the two header column cells by holding down the Control key and clicking each cell. Next, open or expand the Properties window. The Properties window contains a Border Style property for each area of the selected cells—top, bottom, left, and right. For this example, select Solid for the bottom border.

With the Properties window still open, click the Estimated Cost detail row cell. Format the cell in the Properties window to make it currency by adding the formatting command C0 for the Format property.

Next, rename the header cell labeled Employee Name to Diagnosis > Service > Patient > Employee and extend the column to accommodate the width of the new label.

After you apply the formatting, you can immediately see how these changes affect the output by clicking the Preview tab (see Figure 6-7).

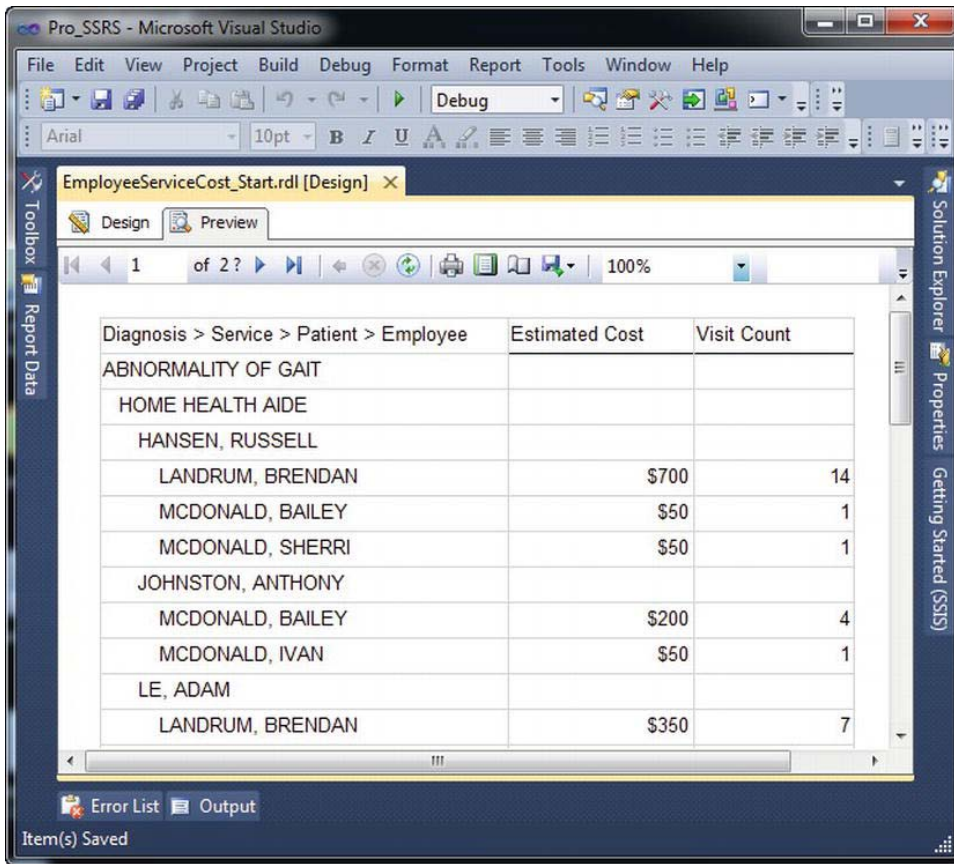


Figure 6-7. Report output with formatting

The EmployeeServiceCost_Format.rdl report in the Pro_SSRS project has the formatting elements applied.

Adding Subtotals

Having subtotals at each grouping level makes the report much easier for the user to read. This is especially true if the report has interactive drill-down features, as yours will have.

Adding subtotals to the groups is as easy as copying and pasting or entering the field value expressions—in this case, `=Sum(Fields!Visit_Count.Value)` and `=Sum(Fields!Estimated_Cost.Value)`—to the upper level grouping cells in the report. For this report, you have defined a group for the Employee_Name field in the detail row. This forces the report to calculate the sums of the Estimated_Cost and Visit_Count fields for each employee. For this report, this is all that is required, but for many other reports you will be creating, you will need to include the detail records for a more granular analysis.

In preparation for adding interactivity to the report, you will sum the `Estimated_Cost` and `Visit_Count` fields at all grouping levels by copying the Employee Cost and Visit Count textboxes at the Employee Name group level and pasting them into the cells in each group heading row—in this case, for Diagnosis, Service Type, and Patient Name. Alternatively, you could choose to enter the Sum value expression or to select Sum under the Summarize By submenu, as mentioned previously. If you choose one of these methods rather than copying and pasting, you will need to apply the currency formatting (C0) as you did previously. You will also make the topmost grouping (the Diagnosis field) bold by holding the Control key and clicking to highlight each of the Estimated Cost and Visit Count values in the Diagnosis group and then clicking the Bold button on the toolbar. With the bold formatting applied, the summed values at the group level will be easy to distinguish from the detail row values. The design of the report should look like Figure 6-8.

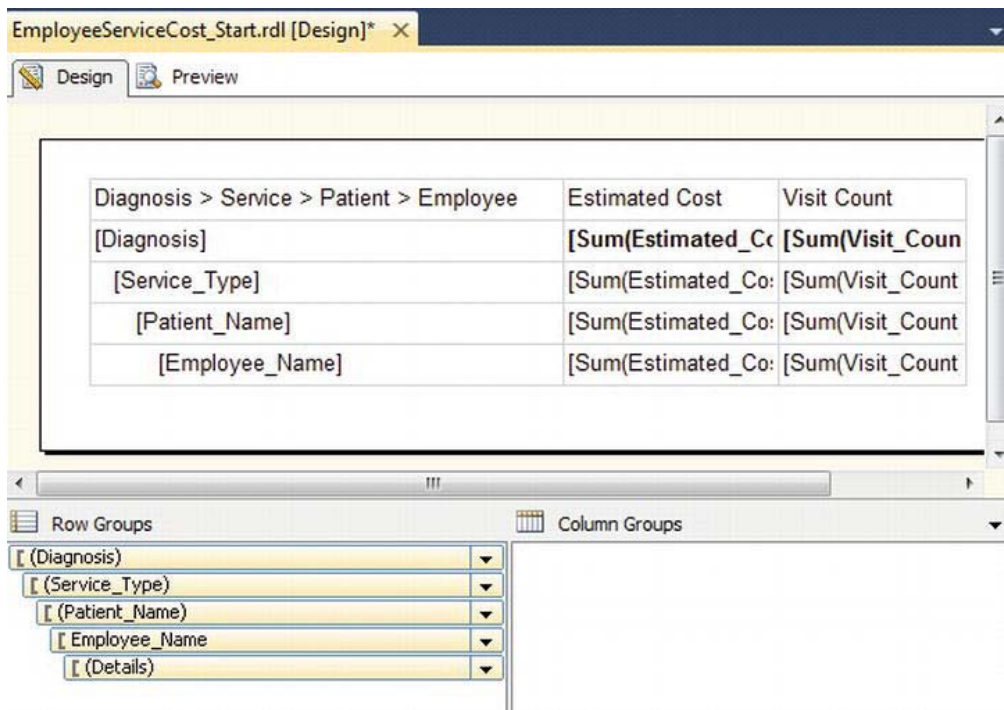
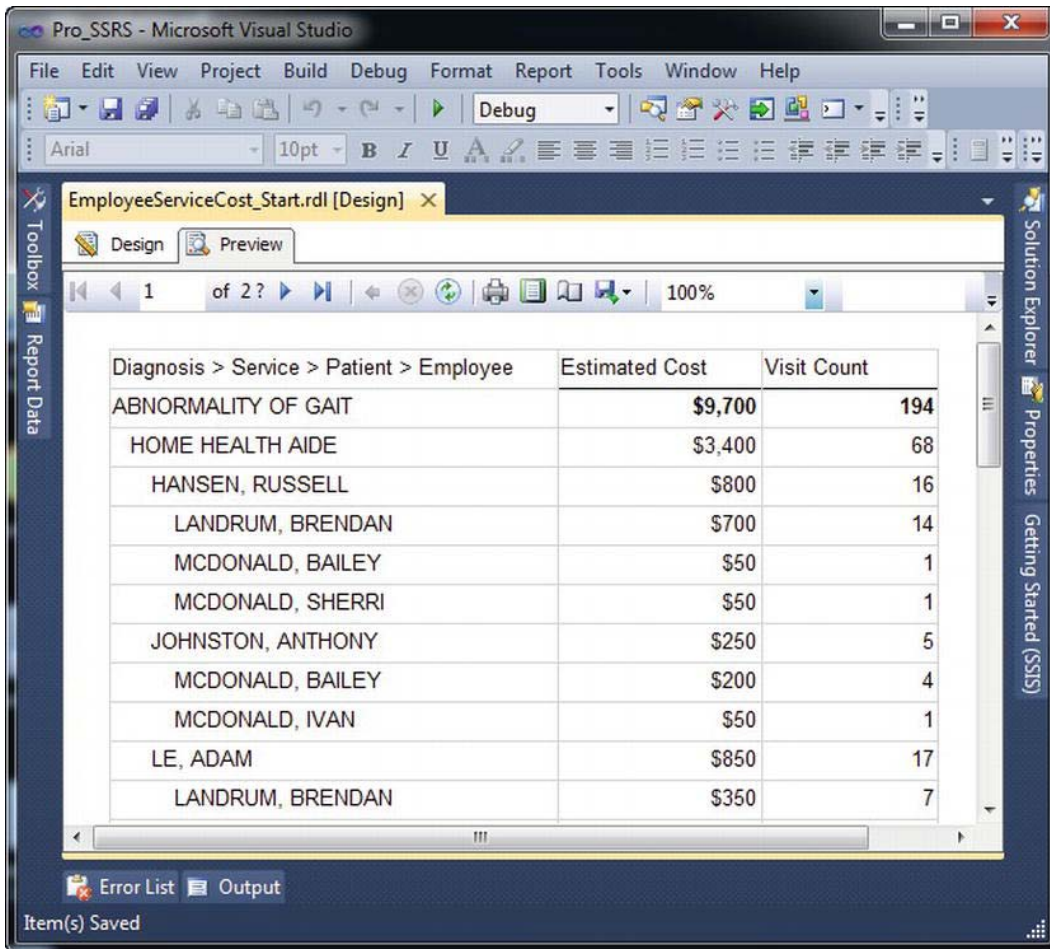


Figure 6-8. Report design with grouping level subtotals

The output of the report, which you can see by selecting the Preview tab, has much more valuable information now for each grouping. For Abnormality of Gait's disease, for example, you can now see that there were 194 services for a total estimated cost of \$9,700, and the bold formatting helps visually separate the values. The patient Russel Hansen (whose name is indented because of the padding you applied to the Patient group) is an Abnormality of Gait patient and has had 16 of the 68 home health aide visits. You can further see each employee's visit count and the cost for this patient in the rows containing the Employee_Name grouping (see Figure 6-9).



Diagnosis > Service > Patient > Employee	Estimated Cost	Visit Count
ABNORMALITY OF GAIT	\$9,700	194
HOME HEALTH AIDE	\$3,400	68
HANSEN, RUSSELL	\$800	16
LANDRUM, BRENDAN	\$700	14
MCDONALD, BAILEY	\$50	1
MCDONALD, SHERRI	\$50	1
JOHNSTON, ANTHONY	\$250	5
MCDONALD, BAILEY	\$200	4
MCDONALD, IVAN	\$50	1
LE, ADAM	\$850	17
LANDRUM, BRENDAN	\$350	7

Figure 6-9. Report output with grouping level subtotals

The EmployeeServiceCost_Subtotals.rdl report in the Pro_SSRS project includes subtotals.

Adding Interactivity

Regardless of the audience for a particular report—whether it is a decision maker interested in on-screen summarized data or a knowledge worker who needs the ability to print reports—interactivity within the report makes navigating to specific information easier and more efficient. You can provide interactivity within an SSRS report in several ways. You will be working with four basic types of interactivity in the following sections:

- *Document mapping*: Provides a navigation pane within the report with values based on a field or grouping.
- *Visibility*: Adds interactivity to a report by hiding and showing report items based on user input.
- *Interactive sorting*: Allows users to interactively select how the report data will be sorted.
- *Hyperlink actions*: Allows the user to click a report item that is linked to a location within the same report or external to the report.

The different rendering formats provided with SSRS, which are covered in detail in Chapter 7, accommodate viewing and printing reports to meet the needs of different types of workers. However, this produces a limitation in that some of the functionality of one rendering format is not available in other formats. This is most evident when working with interactivity, as you will see in the “Interactive Sorting” section.

Document Mapping

Creating a document map in an SSRS report will present users with an integrated navigation pane when the report is rendered. The user can select an item in the navigation pane, which will cause the report to jump to the position where that item is located. In the example report, for instance, a user might be interested in viewing information about Alzheimer’s patients. You can create a document map for the Diagnosis group in the report so that when the user selects Alzheimer’s from the navigation pane, the report will automatically skip to that section; in other words, the user will not have to manually search through the report to find the desired information. You can also add document maps at multiple levels, creating a hierarchical selection in the navigation pane. Keeping with the example, you can add a document map to the Service_Type group in addition to the Diagnosis group; the user can then expand Alzheimer’s in the navigation pane to see all the types of services—home health aides, for example—that have been performed for each diagnosis.

You create document maps by adding an expression to the Document Map Label property available for individual report items or for groups. Begin this example by opening the EmployeeServiceCost_DocumentMap_Start.rdl report in the Pro_SSRS project. By following these steps, you will add a document map label to the Service_Type and Diagnosis groups:

1. On the Design tab, right click the Service_Type group in the Row Groups pane and select Group Properties. Next click on the Advanced tab.
2. In the Advanced tab of the Group Properties, select Service_Type in the Document map dropdown.
3. Complete steps 1 and 2 for the Diagnosis group, which is the first-level grouping above Service_Type. Select Diagnosis for the Document Map Label option.

Now when you preview the report, the navigation pane will automatically be displayed on the left side of the report. The preview, which is in HTML by default, displays in one of the rendering formats that supports document mapping, such as PDF (see Figure 6-10).

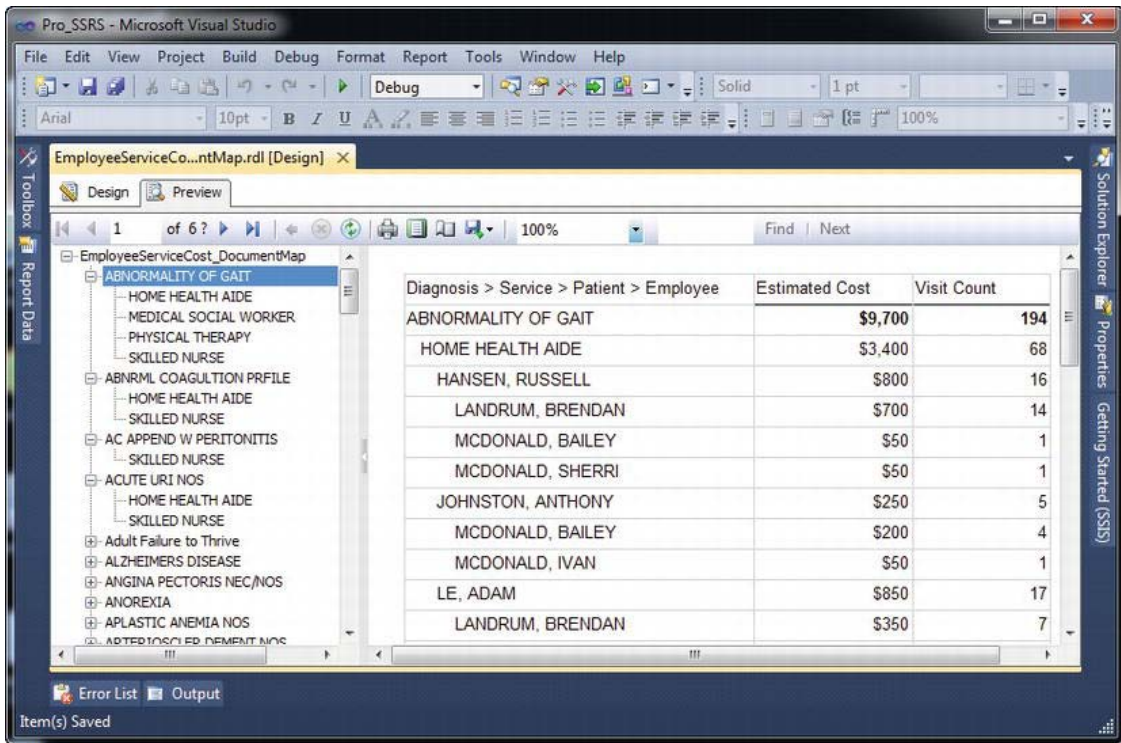


Figure 6-10. Report with navigation pane from document map

The EmployeeServiceCost_DocumentMap.rdl report in the Pro_SSRS project includes a document map.

■ **Note** Adobe Acrobat views document mapping as bookmarks when the report is rendered in PDF. Bookmarks in SSRS perform a different function altogether. They are embedded within the report, and report items are assigned bookmark links.

Visibility

Another feature of SSRS is the ability to show or hide areas of the rendered report based on user input, results returned in a dataset or based on some predefined logic. Often users want to see only summary information on a report, but would like to be able to drill into the summary data to see the detail information if necessary. Report designers might make two reports, a summary and a detail report, which have to be updated and maintained separately. These reports are often based on the same query. Fortunately, SSRS's ability to show or hide report data does away with the need to create separate reports. The visibility properties for report items control which report items are shown or hidden.

Let's assume you have distributed your report to your intended audience, and they have come back with "suggestions" for how to improve the report—this is real world reporting, after all. They indicate that they would like to see the following:

- Summary totals for the visit count and the estimated cost of each diagnosis when the report is first rendered, but with the ability to drill into the details of the patients and employees if warranted.
- The number of patients who have a specific diagnosis.
- The number of individual employees who have provided care for these patients.

With SSRS, this is fairly straightforward, and you can knock out an improved report quickly. Start this example by opening up the `EmployeeServiceCost_Visibility_Start.rdl` report in the `Pro_SSRS` project. Then, you'll just need to follow these design steps before modifying the visibility properties:

1. On the Design tab, right-click the entire column labeled **Diagnosis ► Service ► Patient ► Employee**, and select **Insert Column ► Inside Group - Right**. Complete this step one more time so that we have two empty columns between the groups and the Estimated Cost and Visit Count columns.
2. Enter **Employee Count** and **Patient Count** as the new column header text in the second and third columns, respectively.
3. Resize the second and third columns in the table from right to left so that they are approximately 1 inch each. If, by default, you do not see the ruler, you can right-click a blank area of the design environment and select **Ruler** under the **View Menu**.
4. Highlight every cell in the **Service Type**, **Patient Name**, and **Details** rows. You can accomplish this by holding down the **Control** key and clicking the row marker to the left of the first column in the table. Once all the rows are highlighted, select an 8-point font size from the formatting toolbar.

You can control the visibility state of report items, hidden or visible, by setting the visibility property values. You can hide report items at any level in the report and toggle their visibility property values when a user clicks the + or – icon to show or hide them. The toggle point of the hidden items is another report level, such as a group. In this example, you would like to hide every level except the **Diagnosis** and **Service_Type** fields, but give the user the ability to show or hide the details. To begin, hide every group except **Diagnosis** and **Service_Type**. The steps to accomplish this are as follows:

1. Right-click the **Employee_Name** group in the **Row Groups** section and select **Group Properties**. Then click the **Visibility** tab.
2. On the **Visibility** tab, select the **Hide** radio button.
3. Enable the checkbox to **Display Can Be Toggled by This Report Item**.
4. In the **Report Item** drop-down list, select **Patient_Name**.
5. Perform steps 1 through 4 for the **Patient Name** group, selecting or typing **Service_Type** as the toggle report item.

The other two requests were to be able to see the patient and employee totals for each diagnosis. You can add an expression, CountDistinct, to the report that will count each unique patient and employee and calculate the amounts at the diagnosis level. The syntax used for the patient count is as follows:

```
=CountDistinct(Fields!FieldName.Value)
```

By adding the CountDistinct expression for the field PatID (which you know to be unique per patient) as well as for the field EmployeeID, it will be much easier to see at a glance how many patients with a specific diagnosis have received care.

Place the following two expressions for the Diagnosis group in the cells just below the Employee Count and Patient Count header cells:

```
=CountDistinct(Fields!EmployeeID.Value) =CountDistinct(Fields!PatID.Value)
```

Though the report is still similar to the non-interactive report, with the drill-down additions it will look much different when previewed (see Figure 6-11). In the screenshot in Figure 6-11, you will notice that we hid the document map to provide more space for the report. You can show or hide the document map by double clicking the tiny arrow in the center of the bar separating the document map and the body of the report.

Diagnosis > Service > Patient > Employee	Employee Count	Patient Count	Estimated Cost	Visit Count
ABNORMALITY OF GAIT	27	15	\$9,700	194
HOME HEALTH AIDE			\$3,400	68
MEDICAL SOCIAL WORKER			\$200	4
PHYSICAL THERAPY			\$3,900	78
SKILLED NURSE			\$2,200	44
ABNRMAL COAGULTION PRFILE	29	15	\$11,050	221
HOME HEALTH AIDE			\$6,150	123
SKILLED NURSE			\$4,900	98
AC APPEND W PERITONITIS	4	1	\$400	8
SKILLED NURSE			\$400	8
ACUTE URI NOS	9	2	\$1,200	24

Figure 6-11. Report with interactive drill-down

The `EmployeeServiceCost_Visibility.rdl` report in the `Pro_SSRS` project includes the visibility properties.

Interactive Sorting

It never fails when deploying a report to a large audience: someone will ask that the report be sorted in a certain way that usually differs from the way it was originally designed. When this scenario happens, typically the report designer is torn between creating a second, almost identical, report with custom sorting to appease the requestor and placing the request in queue for a future enhancement to the report. Interactive sorting allows users to sort the report at runtime on any number of fields that have been defined to use this functionality.

In the sample report, you know that you have a broad audience that may use this report for different purposes. A chief financial officer (CFO), for example, may want to view the report to see which diagnosis has the most number of visits. In contrast, another user may need to understand how many patients have a certain diagnosis and would like the report sorted by patient count, rather than visit count, at the Diagnosis group level. In this section, you will add interactive sorting to the report to meet these two needs, knowing that it is possible to sort the report using any other criteria if requested without having to create additional reports based on a user subdivision.

Since you know that the interactive sorting you will apply to your report will be patient count and not visit count, all that you have to do is add this criterion to each of the textboxes where the users will click to change the sorting based on their needs. Using the `EmployeeServiceCost_InteractiveSort_Start.rdl` as our starting point, add the interactive sorting to the header cells Patient Count and Visit Count by following these steps:

1. On the Design tab, right-click the Patient Count header textbox, and select Textbox Properties. Click the Interactive Sorting tab.
2. Check the box titled Enable Interactive Sorting on This Text Box.
3. In the Choose what to sort area, select the Groups radio button and add Diagnosis as the group expression. In the Sort by area enter `=CountDistinct(Fields!PatID.Value)`.
4. Check the option Apply This Sorting to all Groups or Data Regions in and then select the `Emp_Svc_Cost` data region, or type in that region name. Click OK
5. Right-click the Visit Count header cell, and perform steps 1 through 4, replacing the sort expression with `=Sum(Fields!Visit_Count.Value)`, as shown in Figure 6-12.

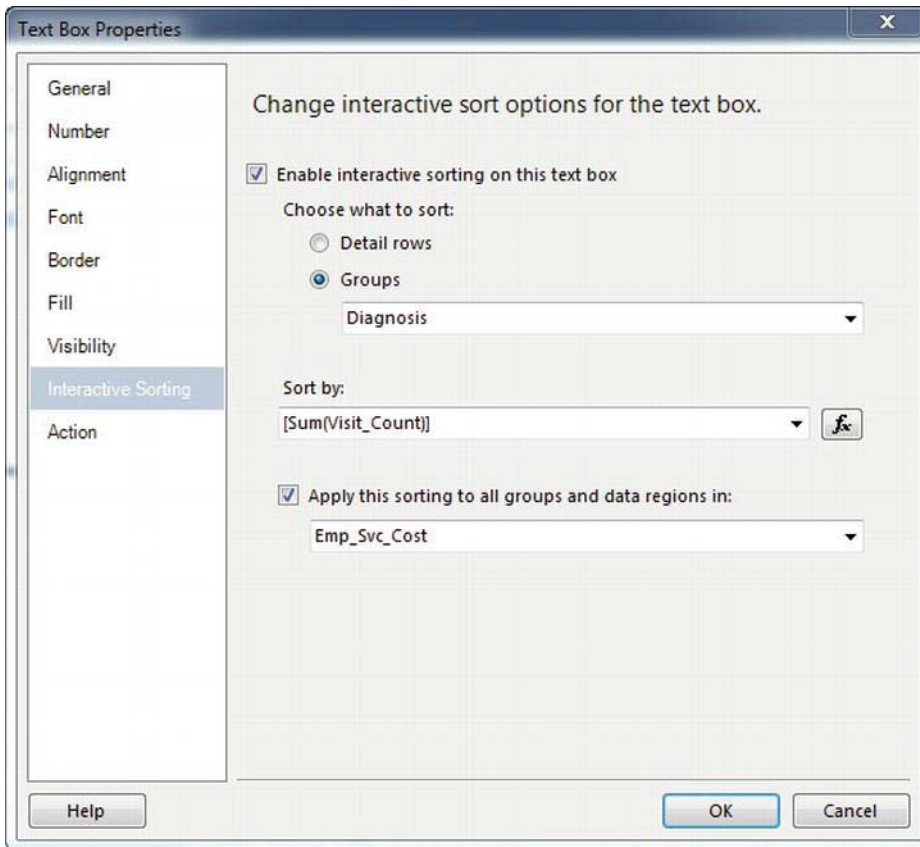


Figure 6-12. Interactive sort properties for the visit count

When you view the report with the new interactive sorting expressions in place, you can see the automatic addition of a selectable sort icon in the Patient Count and Visit Count header fields, as shown in Figure 6-13. When a user clicks this icon in the browser, the report will automatically re-sort to show either the greatest or least number of patients per diagnosis or the greatest or least number of visits per diagnosis. Figure 6-13 shows the diagnosis with the most number of visits, which is Physical Therapy NEC with 1,579 visits and 76 unique patients diagnosed with this illness. The user could also choose to sort the report in ascending or descending order by the number of patients.

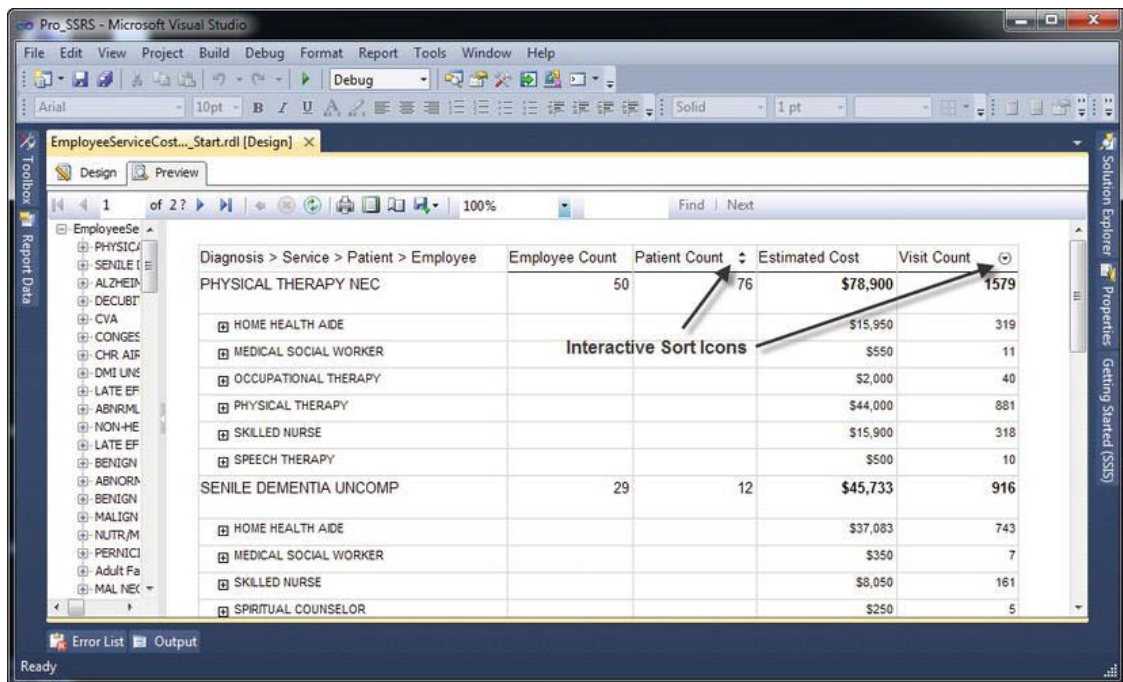


Figure 6-13. Report sorted interactively to show the number of patients

Interactive sorting is a property of the textbox report item and is typically used on column headings in a Table or Matrix data region. A single textbox can control sorting for one or more data regions as long as they are within the same scope or grouping. For example, it is possible to sort multiple tables that are nested within a List data region.

The EmployeeServiceCost_InteractiveSort.rdl report in the Pro_SSRS project includes interactive sorting.

Hyperlink Actions

Having the ability to link one report item, such as the contents of a textbox, to another report or URL adds another valuable level of interactivity in SSRS. By adding hyperlinks to an SSRS report, users can work with the report as they would an application or a Web page, making their tasks more efficient. The sections to follow show how to add several links or actions to your reports to aid users in linking to other reports and locations, such as a company intranet site. You can associate three basic actions with values in a report:

- Jumping to a bookmark
- Jumping to a URL
- Jumping to a report

You can find these on the Action tab of the report items that support these actions, such as textboxes, charts, and images.

To demonstrate each of these hyperlink actions, we will use a report that is more suited to hyperlink actions than the one you have been designing thus far, which already contains one level of interactivity in the drill-down functionality. The next report, Employee Listing, will provide a simple list of employees, grouped according to their clinical specialty. Two reports are available in the code download for the Employee Listing report. One is provided with only the dataset created so that you can step through the following procedures to create the report. It is called EmployeeListing_Start.rdl. The other report is a complete version called EmployeeListing.rdl.

You will add the three interactive hyperlink actions to the report to deliver the following features:

Bookmark: When the employee name is selected, the report will jump to a bookmarked location within the report that contains more details about the employee, such as the number of patients they have seen.

URL: You will also set up a link to the employee's department web site, based on the employee's discipline or clinical specialty. You will also use a report parameter that you will set up specifically for the purpose of selecting the employee's branch location. When users select a branch location from a drop-down list provided with the report parameter, they will be taken to their own department's intranet site. This functionality is shown for example purposes only and will not actually take you to an intranet website; that is, unless you have intranet sites set up for the branches and subfolders for each, containing the disciplines.

Report: You will add a link to your Employee Service Cost report that will pass an EmployeeID parameter to limit the results of the linked report. You will use the same technique in the Employee Service Cost report that will link to a patient survey form using rich-textbox formatting.

The completed Employee Listing report will contain two Table data regions: one for summary information and one for detailed information about the employee's visits. You will add the hyperlink actions to the summary portion of the report, which will be the first page the user sees. Listing 6-1 shows the query for the dataset that delivers employee information. For this report, you will limit the employees to a known set, as shown in the WHERE clause, to keep the report small. You will also add a date range with two parameters, @DateFrom and @DateTo, which you will create and utilize later. The EmployeeListing_Start.rdl report contains default values for a date range from January 1, 2007, to the current date using the Today() function. This query with sample variable settings can be found under the Queries folder in a script named EmployeeListingQuery.sql.

Listing 6-1. Employee Listing Query

```
SELECT
    RTRIM(E.EmployeeID) AS EmployeeID
    , E.LastName
    , E.FirstName
    , E.EmployeeTblID AS EmpTblID
    , E.EmploymentTypeID AS EmploymentType
    , E.HireDate
    , D.Dscr AS Discipline
    , P.LastName AS patlastname
    , P.FirstName AS patfirstname
    , T.ChargeServiceStartDate
```

```

        , D.DisciplineID
        , P.PatID
FROM
    Trx AS T
    JOIN ChargeInfo AS CI ON T.ChargeInfoID = CI.ChargeInfoID
    JOIN Employee AS E ON E.EmployeeTblID = CI.EmployeeTblID
    JOIN Discipline AS D ON E.DisciplineTblID = D.DisciplineTblID
    JOIN Patient AS P ON T.PatID = P.PatID
WHERE
    (T.ChargeServiceStartDate BETWEEN @DateFrom AND @DateTo)

```

To begin, open the `EmployeeListing_Start.rdl` report. The steps to produce the initial basic report, as shown in Figure 6-14 are straightforward, with only a few pointers needed. First, you will be using a Table data region again, so simply drag the table to the report area on the Design surface. When you add a table, BIDS automatically generates three columns. Add another column to the table. Next, add the following fields onto the detail columns: `EmployeeID`, `LastName`, `HireDate`, and `Discipline`. The employee `Discipline` field references an employee's clinical specialty, such as Home Health Aide or Skilled Nurse. Now that we have a start, let's edit the `LastName` field to make it a concatenated value for Employee Name using the `LastName` and `FirstName` columns separated by a comma (Ex. `LastName, FirstName`). Right click on the `LastName` textbox in the details row and select `Expression`. Because the first and last name fields have been padded with spaces, you will want to use the `RTRIM` function to remove the extra spaces. The expression for the last name text box should look like this:

```
=RTRIM(Fields!LastName.Value) & ", " & RTRIM(Fields!FirstName.Value)
```

Now that you have combined `LastName` and `FirstName` into one, change the header from `Last Name` to `Employee Name`. Make the entire header row of the table **Bold**. Finally, let's change the `Employee` column to look a little more like a hyperlink. Select the `EmployeeID` detail field and then set text to be underlined and the font color to blue.

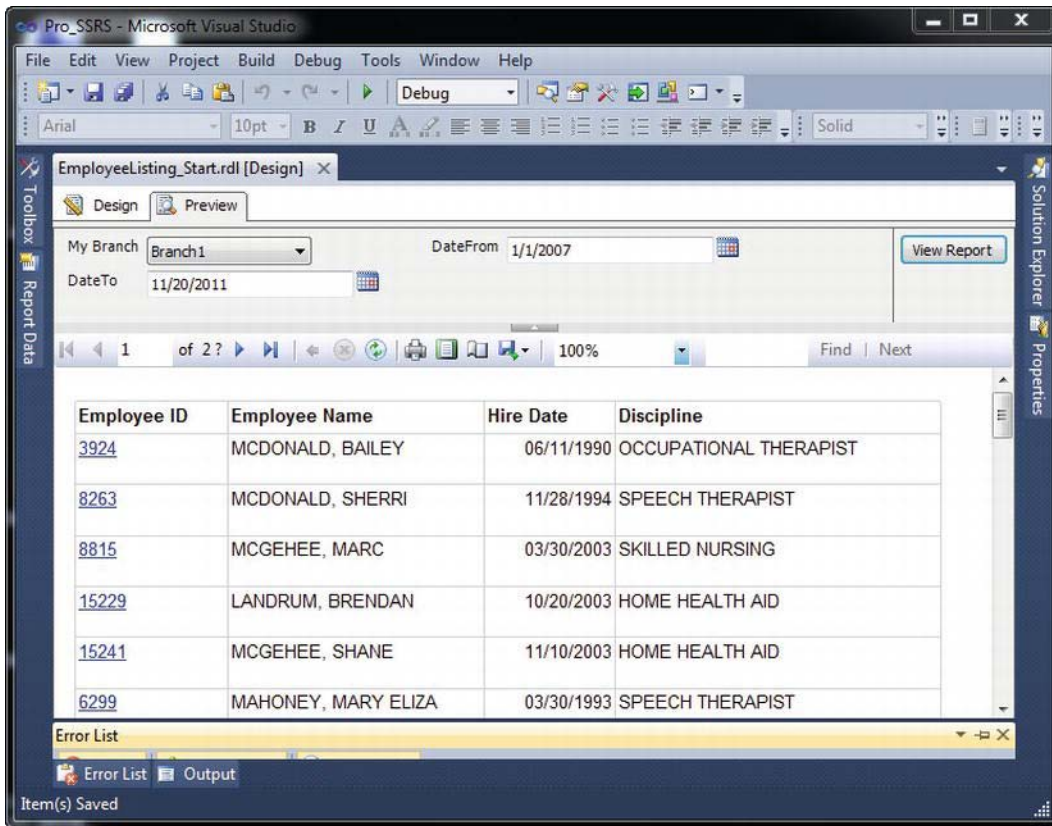


Figure 6-14. Employee Listing report with hyperlink actions

Additionally, when using dates, the default format includes the date and time values, even if there is no time associated with the date. The hire date might look like this, for example: 10/20/2003 12:00:00 AM. By right-clicking and selecting the Textbox Properties of the Hire Date cell, you can change the format from the default to a more standard format. Change the format under the Number tab to MM/DD/YYYY format, such as 10/20/2011, excluding any time value. The format code for this date is d or you can enter MM/dd/yyyy.

Next, because you are returning detail records, with more than one per employee, you need to group the detail row itself using the value of the Employee Name field. You can do this by right-clicking the detail row grouping in the Row Groups area and selecting Group Properties. In the Group On expression field, add the same trimmed employee name as shown in the previous code line. Now, when you preview the report, you have your list of employees to which you can add hyperlink actions. Finally, force a page break after this table so you can add a detailed table that will be used as a bookmark link. To add a page break to a report, simply select the tablix, then right-click a column to get to the table properties, as shown in Figure 6-15. On the General tab, select Add a Page Break After.

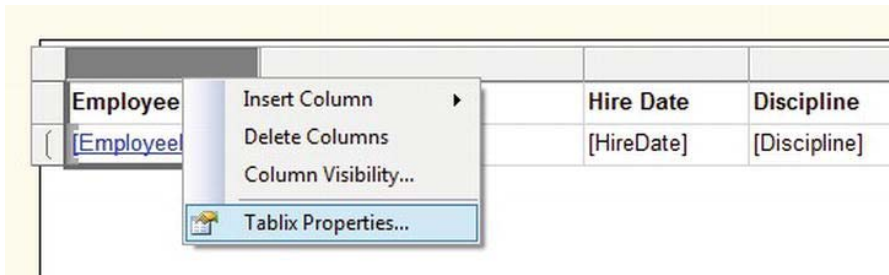


Figure 6-15. Tablix Properties

Adding a Bookmark Link

In this section, you will add a bookmark link to the Employee Name field in the Employee Listing report that, when clicked, will jump to a defined location within the report. In this case, you will not add another Table data region to the report to contain detail information about employee visits. Bookmarks ease the navigation burden on large reports when users are looking for specific information. As discussed previously, summary and detail information can exist within the same report; in the case of adding a bookmark, you are not hiding the data so much as moving it to another location within the same report. The net effect for the user is the same, however, in that they control when they see the detail information.

To add a bookmark to the Employee Listing report, first follow the procedure to drag a new table element to the Design tab. Next, drag the `patLastName` over into the first column of the table. Right-click it and choose Expression. Write an expression similar to the one you did for the Employee Name column earlier, but utilize the `patLastName` and `patFirstName` columns. Rename the header Patient Name. Now, for this table, you will want to group by the employee name, so right-click your detail row and select Add Group and then select Parent Group. You will use the same trimmed expression as the group value expression, as follows:

```
=RTRIM(Fields!LastName.Value) & ", " & RTRIM(Fields!FirstName.Value)
```

Change the name of the group to `Employee_Name`. Then, on the Page Break tab, select the option to enable a break Between each instance of a group. This will force the detail line for each employee to start on a new page. Next, add the date field that represents when the service was performed, `=Fields!ChargeServiceStartDate.Value`, to the third column, and format the date as you did earlier using the `MM/dd/yyyy` style.

Now when you preview the report, the summary employee listing will appear on the first page, and the detail records that show the employee visits will appear on each subsequent page.

Next, you will add a Bookmark property value to the `Employee_Name` field in the detail row of the second table. In SSRS 2012, you will need to add the Bookmark to a field using the Properties window. With the Properties window visible, click the Employee Name field in the details row, shown in Figure 6-16. In the Properties window, enter your trimmed employee name expression into the bookmark value box. This will serve as the pointer record for the bookmark link you will now create.

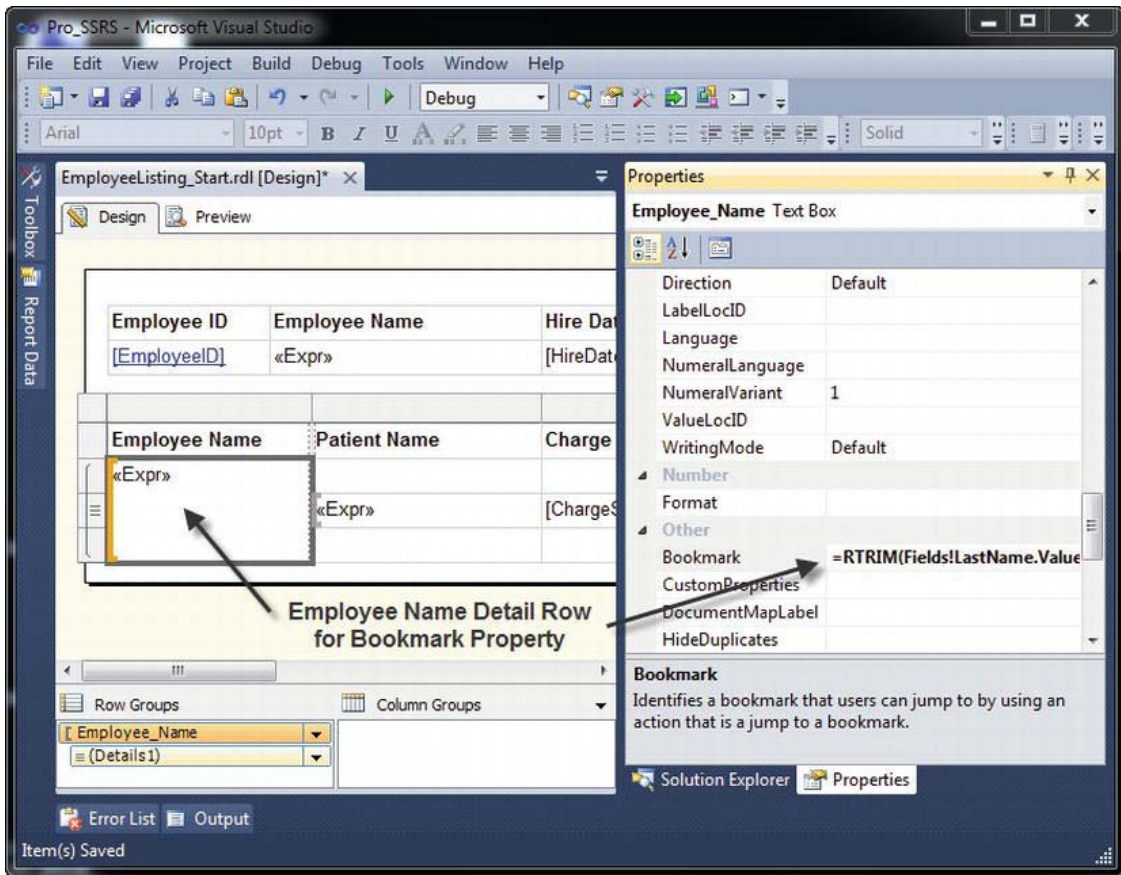


Figure 6-16. Setting a Bookmark Property Value

To create the bookmark link: In the first table, right-click the Employee Name detail row and select Textbox Properties and then click the Action tab. In the Change Hyperlink Options section of the tab, select Go to Bookmark and paste in the trimmed employee name expression you used for the Bookmark property.

■ **Tip** Hyperlink actions do not automatically change the formatting of the field to indicate an associated hyperlink. You can manually change the color and add underline formatting so that the user knows to click the link.

When you preview the report, and click the new bookmark link, you see the detailed information for the selected employee, as shown in Figure 6-17. You should see detailed information for employee

Bailey McDonald; the first three pages of the report are the Employee Listing table, where you clicked the bookmark link in the Employee_Name field.

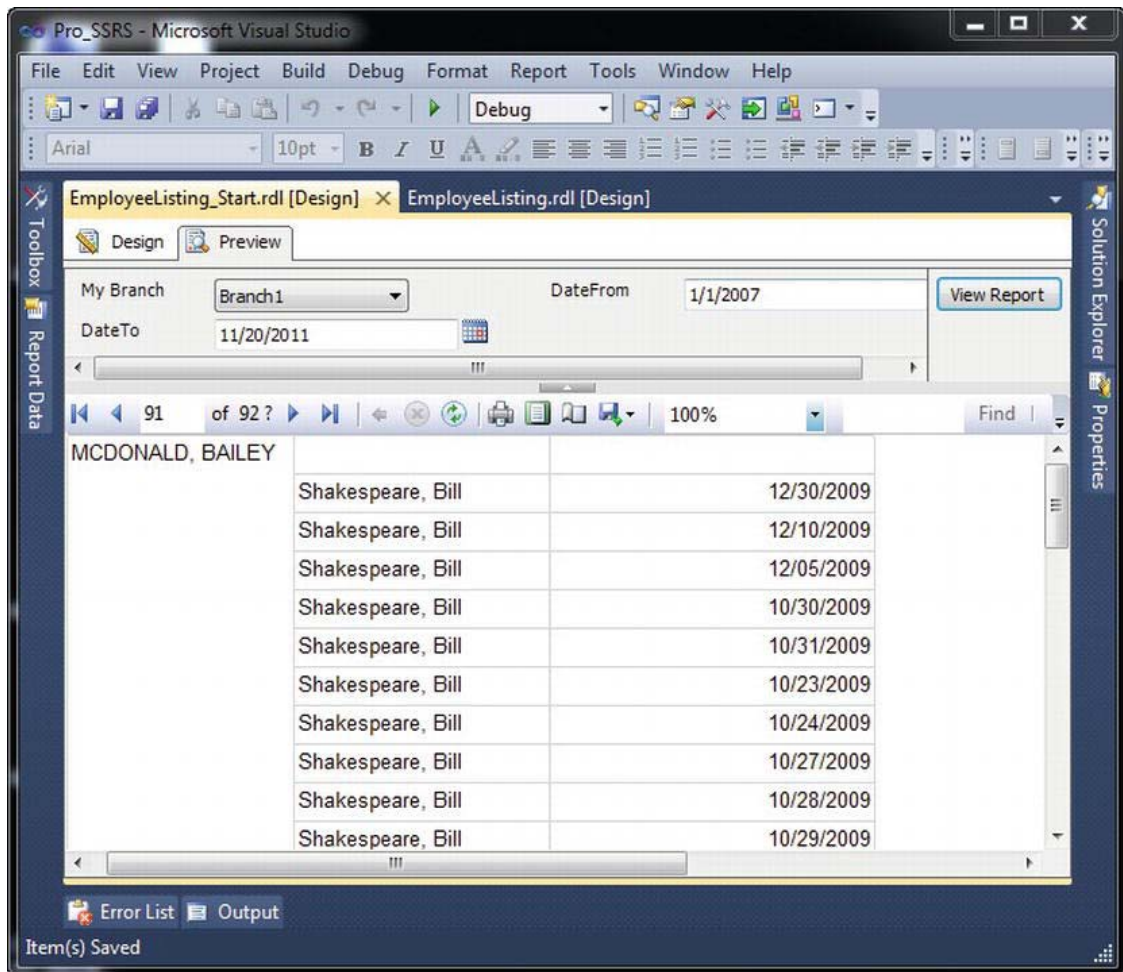


Figure 6-17. Employee visit detail report called from a bookmark link

Adding a URL Link

URL links connect a report to information stored in other locations, such as a Microsoft SharePoint site or the Internet. Like bookmark links, URL links are defined on the Action tab and can be applied to many report items. As discussed in Chapter 4, in almost every value field that is used in SSRS, expressions define the contents. In the case of the URL, you will build an expression that will define the HTTP location, using a combination of the literal URL and a field value from the dataset.

For example, let's assume that your intranet site has a home page that is designed specifically for each employee discipline. An employee who is a home health aide would have a DisciplineID of HHA, and your Web site designed for home health aides would be at `http://webserver1/hha`. Assuming that the same is true for each discipline, it would be easy to add a URL link for each discipline to your report.

Just as you did for the bookmark link for Employee_Name, open the Action tab for the Discipline field. Select Go to URL, and add the following expression:

```
= "http://webserver1/" & Fields!DisciplineID.Value
```

When the DisciplineID field is selected within the report, the browser will open and connect to the site of the specific employee's discipline—for example, HHA for the home health aides site or RN for the skilled nursing site.

Building the URL Link with a Report Parameter

Taking the concept one step further, if you had multiple Web servers at different locations or branches, you would not want to hard-fix the Web server name in the URL string. Using a report parameter to select the server name based on the branch location would make it possible to control the Web server portion of the URL string that you created in the previous example. Let's step through this procedure. Figure 6-18 shows how the Report Parameter Properties dialog box should look.

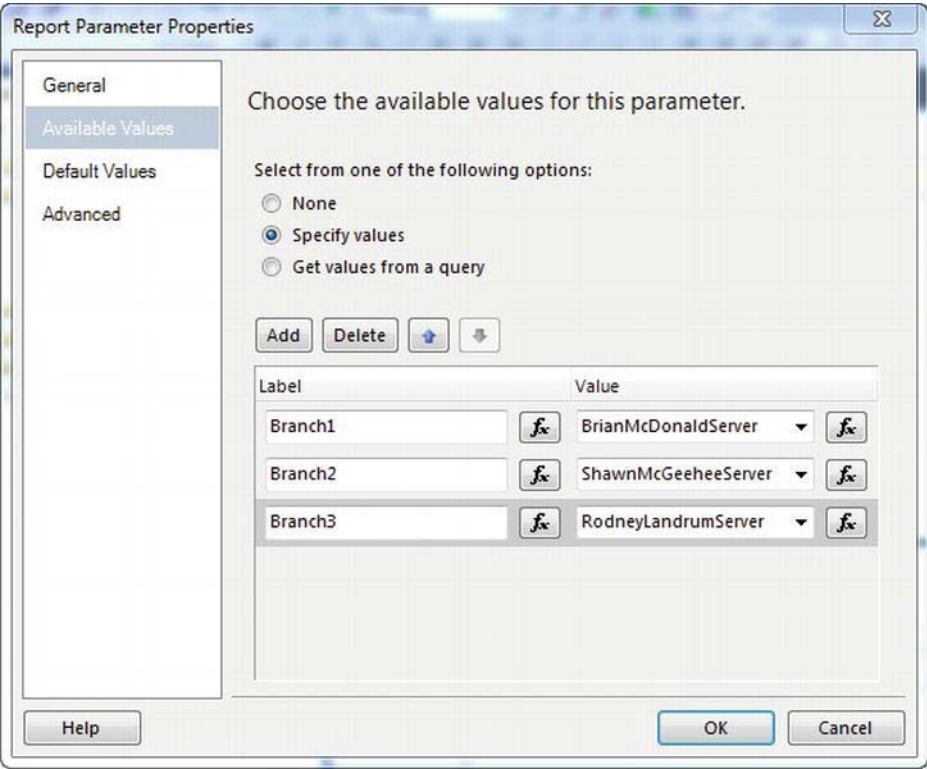


Figure 6-18. Report Parameter Properties dialog box

Follow these steps:

1. While on the Design tab, right-click the Parameters folder in the Report Data pane and select Add Parameter.
2. Enter **Branch_URL** for the parameter name.
3. Enter **My Branch** for the prompt.
4. In the Available Values section, enter the following labels/values: **Branch1** = **BrianMcDonaldServer**, **Branch2** = **ShawnMcGeeheeServer**, and **Branch3** = **RodneyLandrumServer**.
5. Set the default value to be **BrianMcDonaldServer**.
6. Return to the Action tab for the Discipline field, and apply the new expression = "http://" & Parameters!Branch_URL.Value & "/" & Fields!DisciplineID.Value.
7. Preview the report. Notice that you have a new parameter drop-down list called **My Branch** that was set to **Branch1** based on the default value of **BrianMcDonaldServer**.

With the URL location assigned to be that of the parameter **Branch_URL**, whenever a different branch is selected from the drop-down list, the appropriate server for that branch will be selected.

Jumping to a Report

Quite possibly the most useful hyperlink action in SSRS is the ability to link to another report, called a *drill-through report*, from a specified location within the current report. In this section, you will link to a new report, the Patient Survey Letter report, from the Employee Listing report. The Patient Survey Letter report uses a single textbox and demonstrates how you can take advantage of the new rich text formatting made available with SSRS 2008 to form a mail merge style report using both HTML formatting as well as literal text strings, all formatted independently.

You will tie the two reports—the Employee Listing report and the Patient Survey Letter report—together by creating a hyperlink from one to the other. You will also pass a parameter value along with the hyperlink to narrow the results of the Patient Survey Letter report when it is called from the Employee Listing report. The parameter values will be **PatID**, **ServiceMonth**, and **ServiceYear**.

To add the hyperlink action that links to the Patient Survey Letter report, return to the Action tab, this time from the Patient Name detail row textbox within the Employee Listing report. The Patient Name detail textbox is in the second table, shown in Figure 6-19.

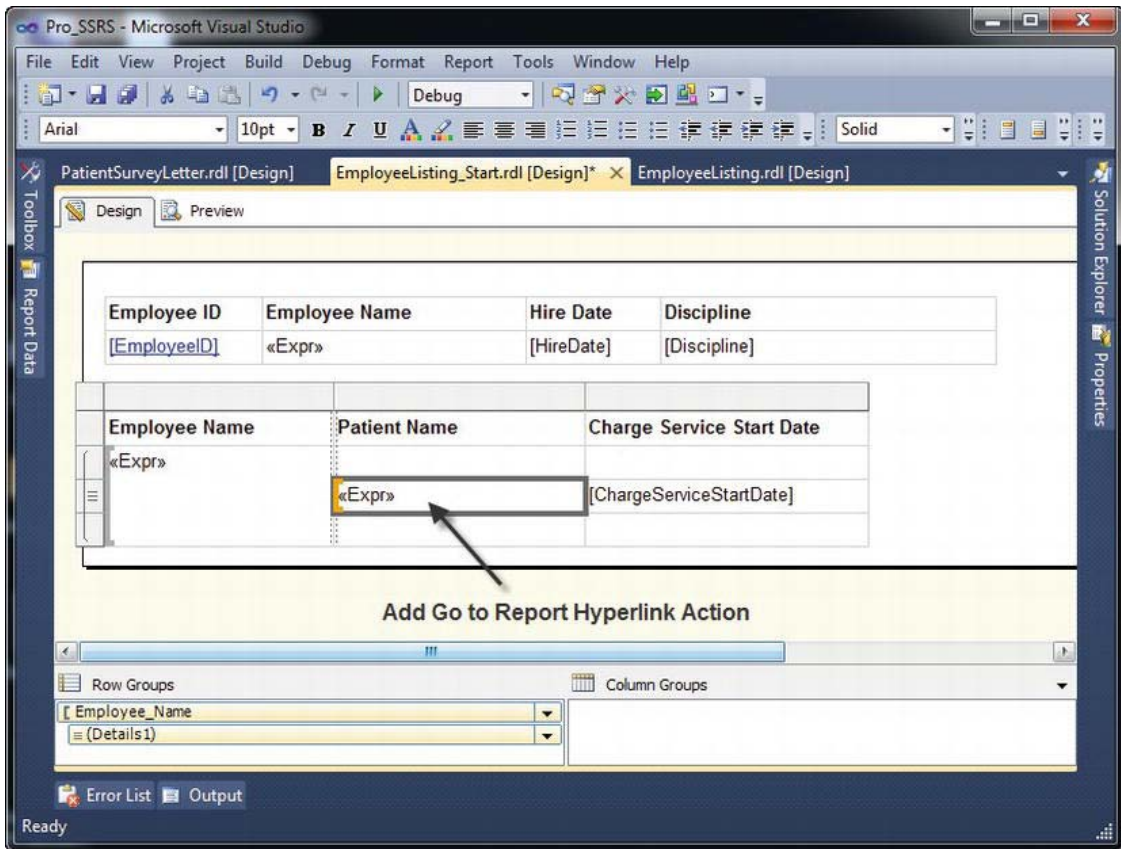


Figure 6-19. Patient Name detail textbox to add Go to Report hyperlink action

After clicking the Go to Report button, a drop-down list appears with all the reports that are available in the current solution. If the report has already been deployed to the report server and is not in the current solution, you can use the relative path based on the target server that is defined in the project. In this example, the target server is `http://localhost/reportserver`. You could add the relative path to any report on the report server. In this case, select the `PatientSurveyLetter.rdl` report, and then click the Add button three times to add `PatID`, `ServiceMonth`, and `ServiceYear`. Choose the parameters that are populated when the report is selected. Later in the “Setting Report Parameters with Stored Procedures” section, we will show how to add these parameters to the report. Choose `PatID` as the parameter, and assign its value as `=Fields!PatID.Value`, which is a field in the Employee Listing report that has a corresponding value to the `PatID` parameter. Next create an expression for `ServiceMonth` as `=MONTH(Fields!ChargeServiceStartDate.Value)` and `ServiceYear` as `=YEAR(Fields!ChargeServiceStartDate.Value)` respectively. After applying the new action, if the Patient Name textbox is clicked when previewing the Employee Listing report, the Patient Survey Letter report will be called and the parameter passed, thus narrowing the dataset for that report to only that for the selected patient.

For the Patient Survey Letter report, you will use a modified version of the Emp_Svc_Cost stored procedure that includes a parameter for the PatID field. The reason for doing this is because the Textbox control does not support filter criteria, and you will need to limit the letter to one patient as it is linked with a single parameter. For merge letters, it is of course possible to use more than one patient. To show both linking to another report and to demonstrate the new textbox formatting, we simply wanted to show the basics of both.

Let's look at the Patient Survey Letter report and the textbox with rich formatting. As you may recall from Chapter 4, textboxes now support placeholders, which can be simple and complex expressions as well as HTML code imported from a table. You will combine elements of each to produce a simple letter addressed to an individual patient. As you can see in Figure 6-20, the letter itself consists of one textbox and one table. The textbox contains a placeholder at its beginning, titled Template_Letter_Header. This placeholder gets its formatting via HTML imported from a dataset that stores the HTML tags in a field.

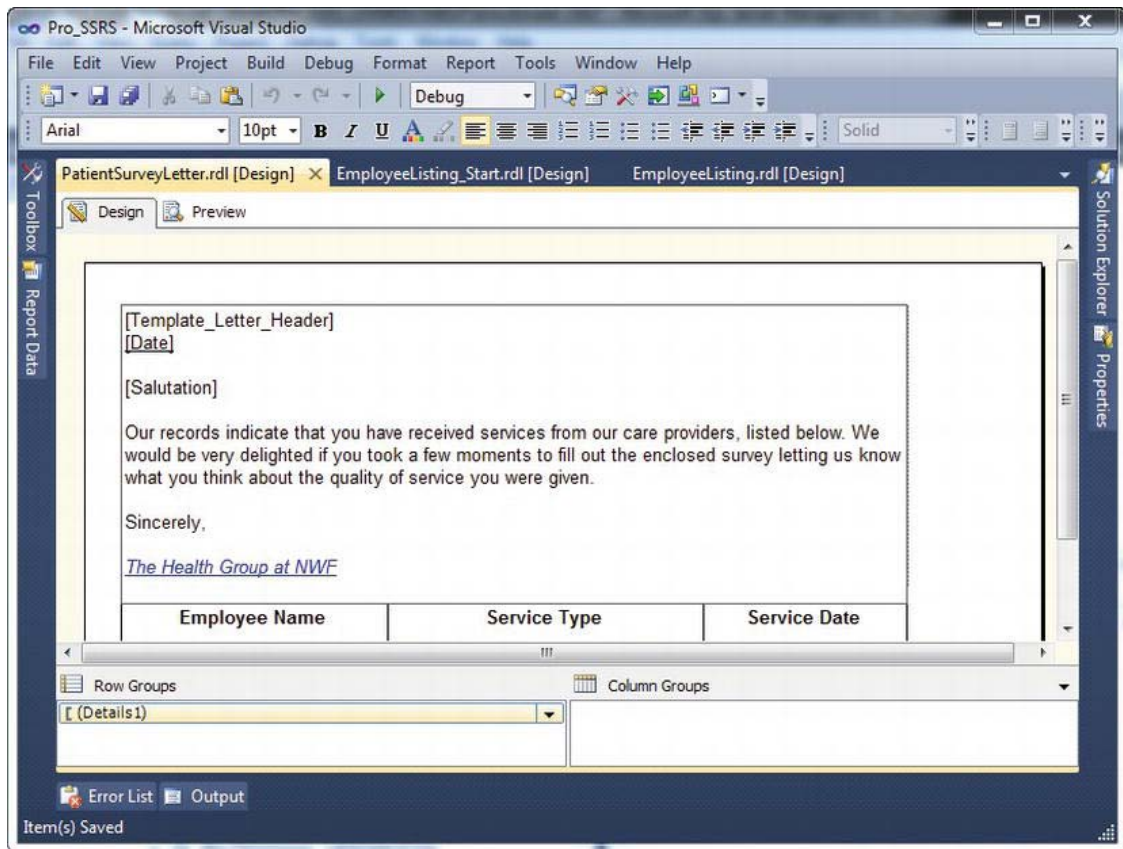


Figure 6-20. Patient Survey Letter report with multiple placeholders

The HTML code is derived from the Format table in the Pro_SSRS database. You can see the sample output of the field in Figure 6-21.

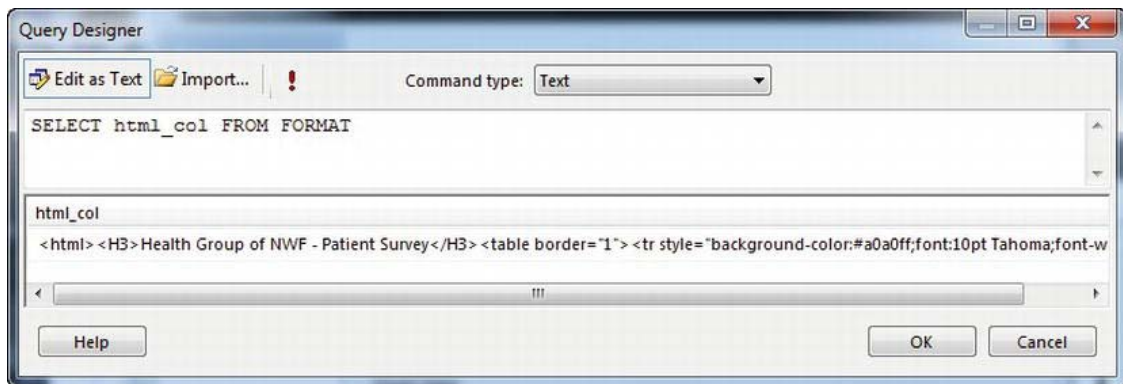


Figure 6-21. Output of the HTML code from the Format table

To assign the HTML code to the textbox placeholder, all that you need to do is right-click anywhere in the textbox and select Add Placeholder. You will label each placeholder as well as assign it an expression value. In this case, you will assign it the value `=FIRST(Fields!html_col.Value, "DataSet1")` as there is only one row in the format table for this example. In order for the HTML to be displayed correctly in the textbox, with all formatting, it is important to select HTML – Interpret HTML Tags as Styles from the Placeholder Properties box, as in Figure 6-22.

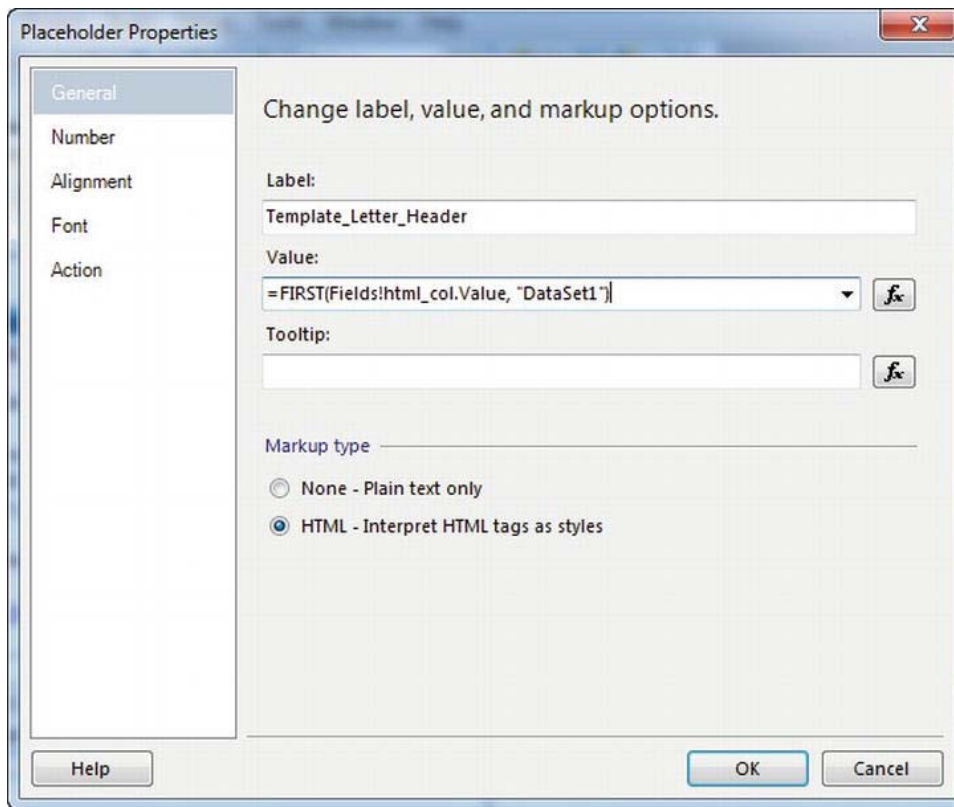


Figure 6-22. Placeholder Properties dialog with HTML selected

The textbox can also contain expressions that correspond to other field values from datasets as well, just as you would expect. After the greeting, you can see that we have added the patient's name as another placeholder. This name, which contains the value expression `=FIRST(Fields!Patient_Name.Value, "DataSet2")`, will change with the parameter value that we set up for the PatID field.

The remainder of the textbox contains additional formatting, such as bold, italicized, and font color of some text, all of which are independent of the header placeholder that contains the HTML code. This kind of formatting was not available before SSRS 2008. With the ability to also export to Microsoft Word, rich text formatting opens up new avenues of report creation for your business. Figure 6-23 shows the rendered Patient Survey Letter report.

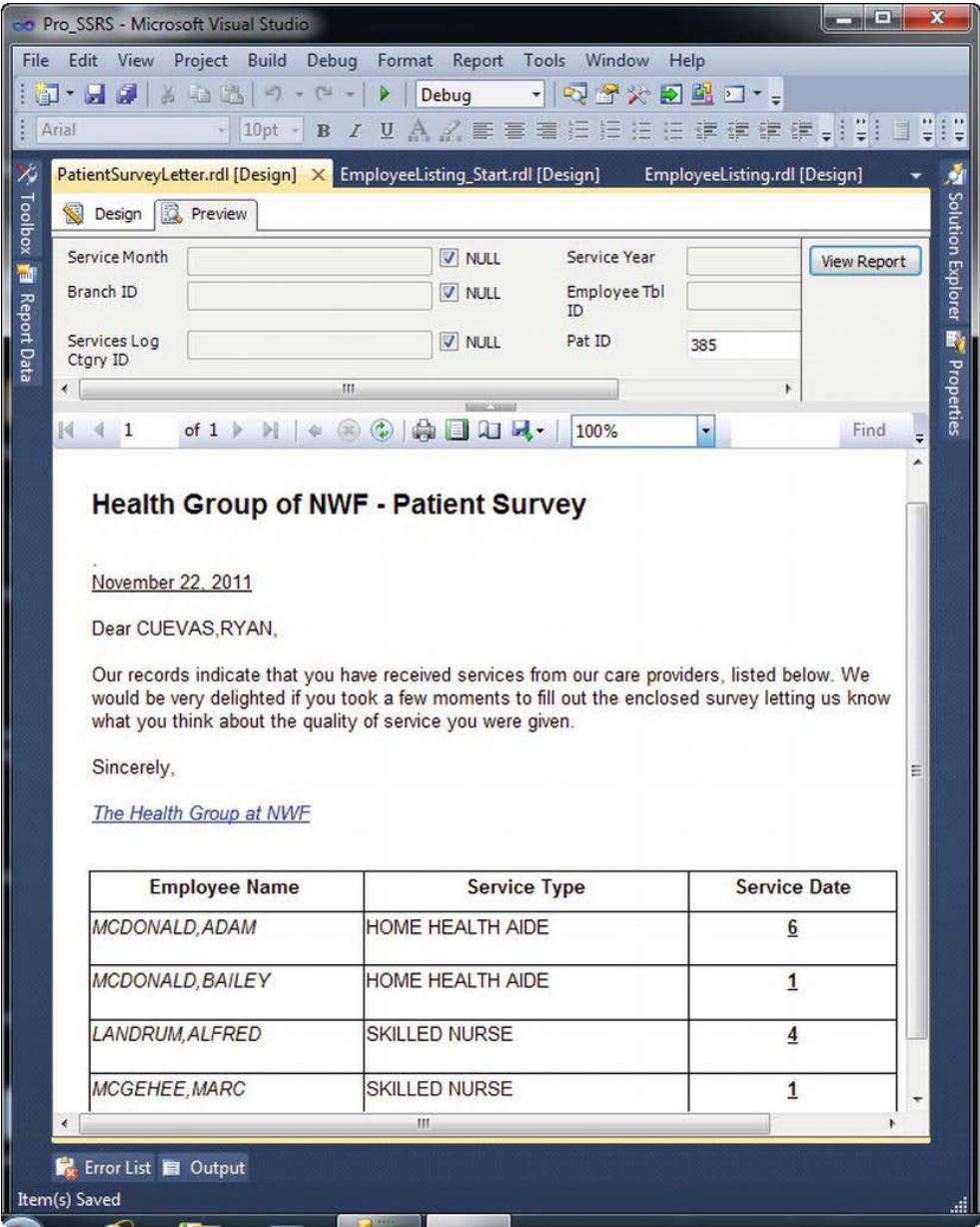


Figure 6-23. Patient Survey Letter rendered

The Patient_Survey_Letter.rdl report is available in the Pro_SSRS project.

Adding Hyperlink Formatting and Tooltips

Before you save the new Employee Listing report with added links, let's add two formatting properties that will make the link more obvious as well as provide feedback on what will happen when the link is selected. The first task is simply to make the EmployeeID field resemble a hyperlink. Select the field, and apply an underline and color format of blue (see Figure 6-24).

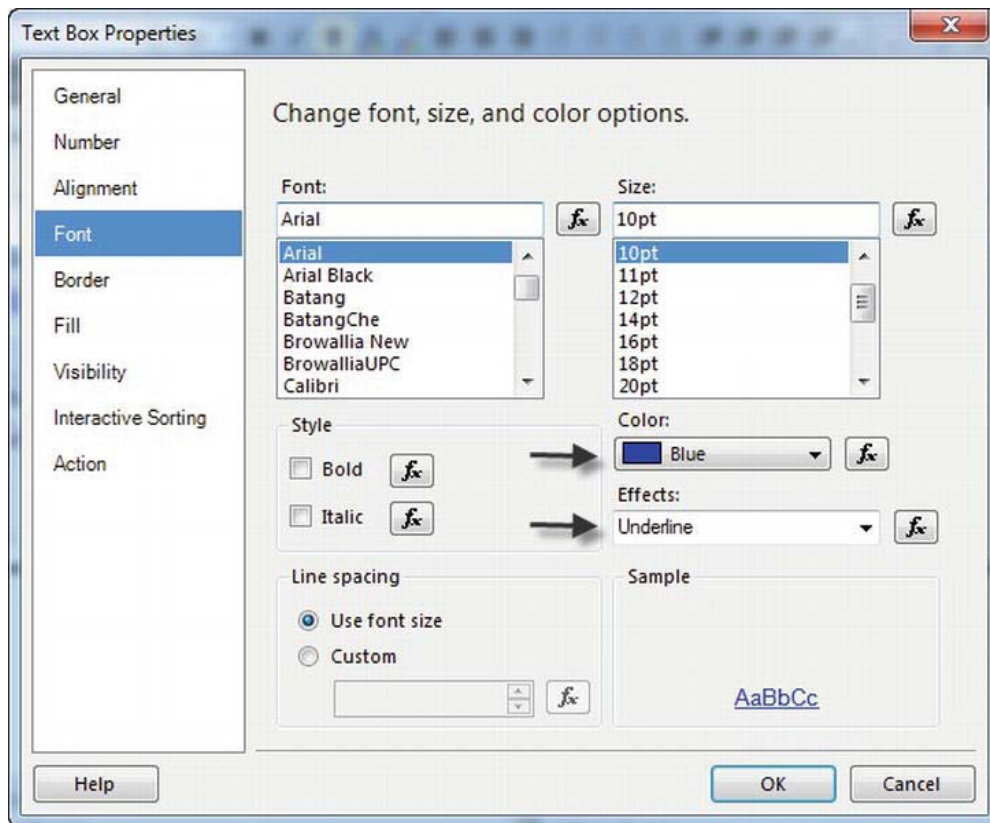


Figure 6-24. Employee Listing report with visible hyperlinks

Next you will add a tooltip to the same field. Tooltips appear whenever a user hover's the cursor over the field, and they provide additional information. In this case, you will use one simply to show which report will be called when the EmployeeID box is clicked—the Patient Survey Letter report. The ToolTip property is located on the General tab of the Textbox Properties window. After selecting the field, open the Properties box, and enter **Patient Survey Letter** as the tooltip. Notice that the tooltip, like most other values, can be an expression as well as a literal string.

It is possible to assign multiple parameter selections, dataset fields, expressions, and fixed literals as input for the drill-through report. Now that you have linked to a report that could potentially have multiple parameters, let's take a closer look at how parameters and filters work together to deliver data to a report.

Setting Report Parameters with Stored Procedures

In Chapter 3, we introduced parameters and explained how you can use them within reports and queries to limit the results returned from the data source. Up to this point, you have been working with different kinds of datasets, queries, and stored procedures to build reports, but we have only touched the surface of how you can use parameters within SSRS. Parameters get their values primarily from user input and are most often associated with a dataset; they are used to limit the amount of data returned. When a parameter is used in this way, it is called a *query parameter*. Query parameters that are part of a dataset, such as a SQL query or stored procedure, automatically generate report parameters within SSRS.

In this section, you will modify the dataset of your Employee Service Cost report to use a parameterized stored procedure instead of a query. By default, report parameters generated from stored procedures do not have populated drop-down lists of data for users to select, so in this section you will also populate the report parameter lists with valid data for user-selectable input. Finally, you will see how SSRS works with NULL parameter values and how to generate a NULL value for the parameter. This will become especially important when retrieving data for your SSRS report, as we will explain later in this section.

You will return to the stored procedure you have already created, called `Emp_Svc_Cost`, which, as you might recall, will deliver the same dataset as the SQL query you have been using. The stored procedure has the added benefit of accepting all the parameters you want to use in the report. SSRS will automatically create the report parameters from the stored procedure. Let's quickly review the parameters that will be passed into the report from the stored procedure:

- BranchID
- EmployeeTblID
- ServiceMonth
- ServiceYear
- ServiceLogCtgryID

To create the parameters automatically for your Employee Service Cost report, which is currently using a non-parameterized query, you will simply change the dataset for your report to the stored procedure.

Open the `EmployeeServiceCost_SP` report from the project included in the code download. In the Report Data window that has the dataset `Emp_Svc_Cost`, you can right-click the `Emp_Svc_Cost` dataset and select Dataset Properties to open the Properties window. In the Dataset Properties dialog, change the Query type from Text to Stored Procedure. Next, select or type the name of the stored procedure, **Emp_Svc_Cost**, in the Query String window, and click OK. When you click OK, the parameters will be created automatically for you. Next, right-click the `Emp_Svc_Cost` dataset again and select Query. You should see that the query will execute the `Emp_Svc_Cost` stored procedure. Click the Run button, which will prompt you to input the parameter values, as shown in Figure 6-25. Since the stored procedure is designed to accept NULL values, change the default input value in the Define Query Parameters dialog box from Blank to NULL, and click OK to complete the execution. If you do not select NULL instead of Blank, the query will fail with an error message, "Failed to convert parameter value from a string to Int32."

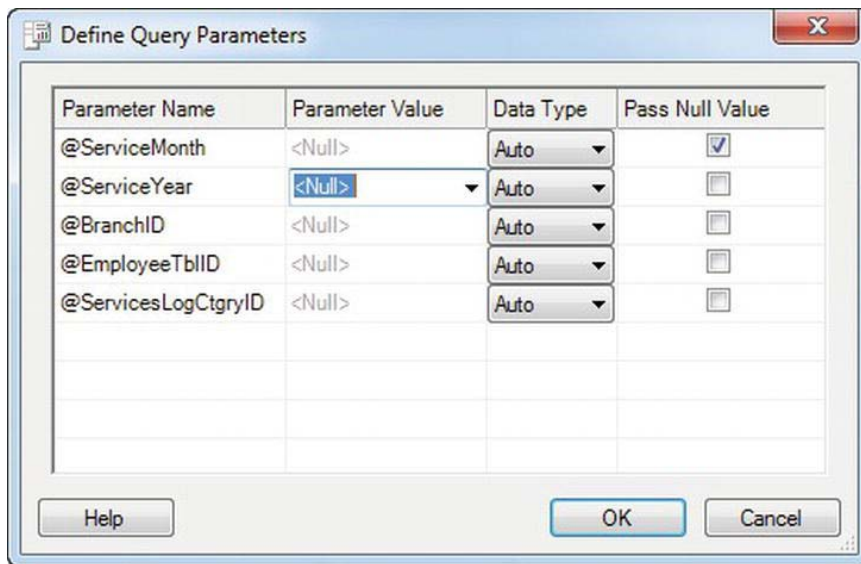


Figure 6-25. Parameters required for the stored procedure *Emp_Svc_Cost*

In the Report Data window, you can see that the report parameters were automatically created from the stored procedure. Although SSRS did correctly assign the datatype for each parameter, integer, and string, it did not automatically set the field to allow NULL values (see Figure 6-26). For the purpose of this report, which expects NULL values as possible parameters, it is important that the Allow Null Value checkbox is selected for each parameter so that when the report is previewed, NULL will be the default value, and the NULL checkbox will be automatically checked so that the report executes without requiring user input. Go ahead and update all of the parameters to Allow null value by right clicking each of them and selecting Properties. Then put a check in the option Allow null value under the General tab.

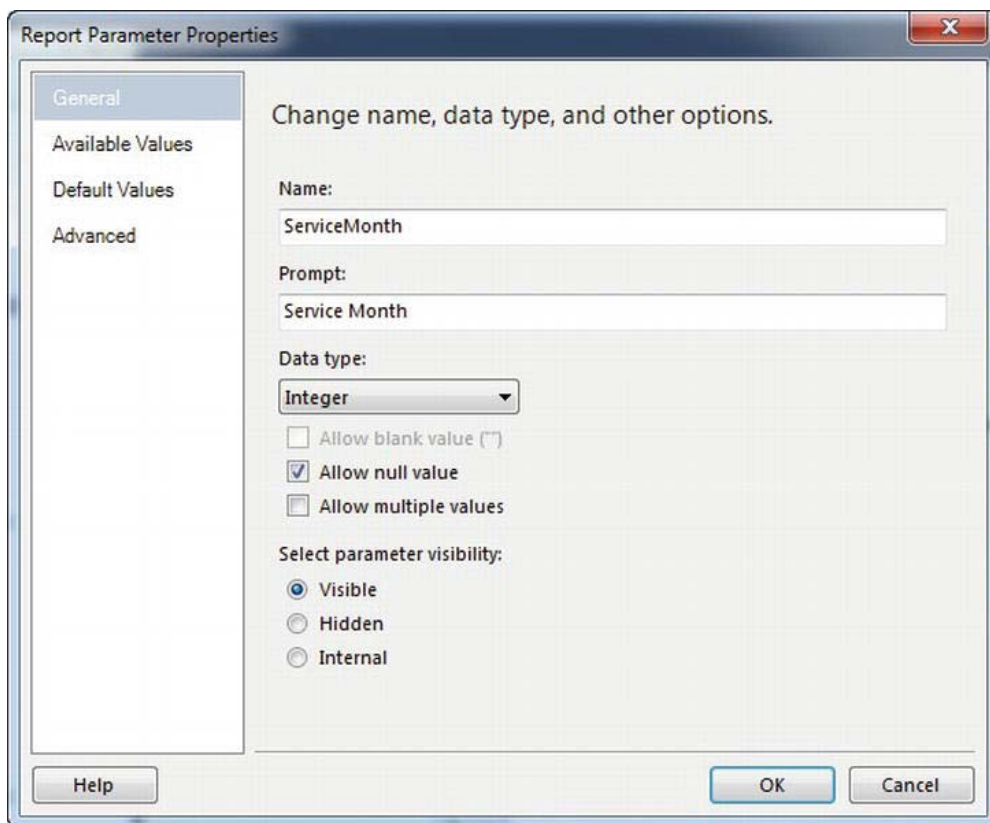


Figure 6-26. Report Properties dialog box, showing the Allow Null Value field

Default parameter values will also need to be manually configured. If no default parameter value is assigned to an available parameter, the report, when rendered or previewed, will not process the incoming data until a user supplies a value. Previewing the report without modifying the parameter selection reveals that the user would need to enter a value for each parameter that has no default value assigned. The user would not be able to choose from a list of values, but would have to enter them manually. This is often unacceptable because the user may not always know the correct values to enter; good examples of this are the EmployeeTblID field that is used to select a specific employee and the BranchID field used to retrieve the branch name.

The first step is to provide valid query-assigned values for the parameter drop-down lists. It would be beneficial to provide a view of the report in preview mode prior to adding descriptive parameter values from a new dataset (see Figure 6-27). Notice that there is a NULL checkbox selected next to the parameter selections. The NULL checkboxes appear when the parameter allows NULL values, as you set earlier, and there are no other available values.

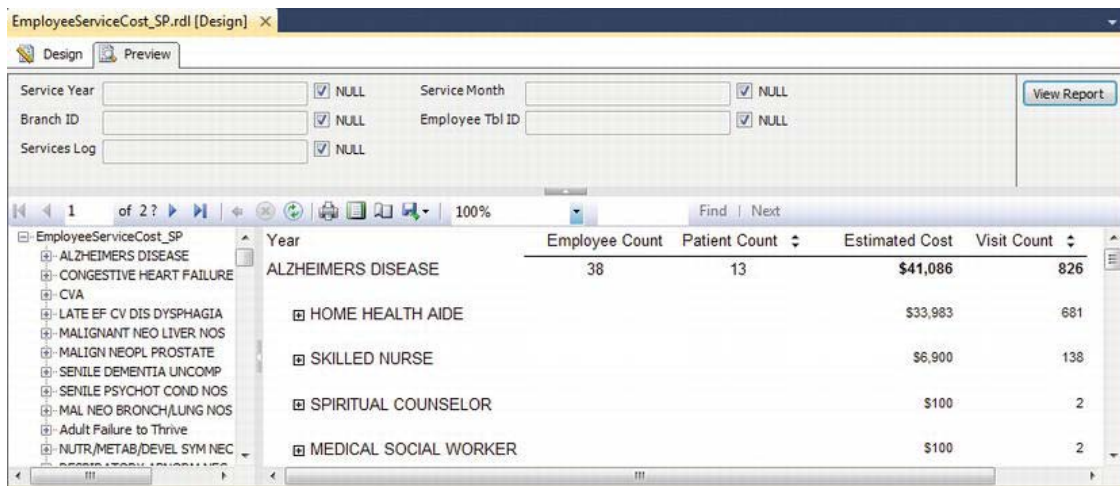


Figure 6-27. No parameter values available via drop-down list

The following scripts can be used to add two datasets to populate the Branch and Employee drop-down lists for the parameters:

1. In the Report Data window, create two new datasets embedded in the report, Employee_DS and Branch_DS, by right-clicking the Datasets folder and selecting New Dataset. For both datasets you create, you will add simple queries that will return the IDs (for the Value attribute) and names (for the Label attribute) for the employee and the branch. Notice in the WHERE clause of the employee query that follows that you are including only a known set of employees for simplicity. In a real world scenario, business rules dictate filters, but you probably wouldn't hard code your values in this fashion. We are just showing one way that you could hard code a filter to return a subset of data to be used in our parameter drop-down list.

--Query for Employee Parameter

```
SELECT
    EmployeeTblID
    , RTRIM(RTRIM(E.LastName) + ', ' + RTRIM(E.FirstName)) as Employee_Name
FROM
    Employee E
WHERE
    (E.EmployeeTblID IN (32, 15, 34, 44, 129, 146, 159, 155, 26))
```

--Query for Branch Parameter

```
SELECT
    BranchID, BranchName
FROM
    Branch
UNION
SELECT
    NULL AS BranchID, NULL AS BranchName
```

2. After you have created the datasets with the previous queries and verified that they execute properly via Query in the Report Data window, expand the Parameters folder. Right-click the BranchID parameter, select Parameter Properties, and enter **Branch** for the prompt for clarity. You will be selecting the branch name in the drop-down list.
3. In the available values for the branch parameters, select Get Values from Query, and then select the Branch_DS dataset. The Value field will be BranchID, and the Label field will be BranchName.
4. Follow the same steps to modify the Employee parameters, assigning Employee_DS and choosing the Value and Label fields as EmployeeTblID and Employee_Name, respectively. Finally change the prompt to **Employee** for clarity as we performed in step 2. When finished, select OK.
5. Finally, on the Design tab, you will add a grouping for Branch Name to the table in the report so that as the parameters are selected, you can see that the report is specific to a branch. To do this, right-click the row header to the left of the Diagnosis textbox in the table, and select Add Group >> Parent Group. This will make the Diagnosis group, formerly the first group, now the second group and will add a new group. Assign the expression value of =Fields!BranchName.Value to the new group, select the option to Add group header and click OK in the Grouping dialog box. Next, move the BranchName field from the new first column row for the Branch group you just created and into the blank area just above Diagnosis. Also, make the field bold, and resize the font to 12 points. Now that it is formatted, delete the column that was created when you added the new BranchName group. You can delete the column by right clicking it and selecting Delete Columns.

The report will now have populated drop-down lists for the available parameter values, as shown in Figure 6-28. Notice that for the two parameters where you have added available values, the NULL checkbox has disappeared.

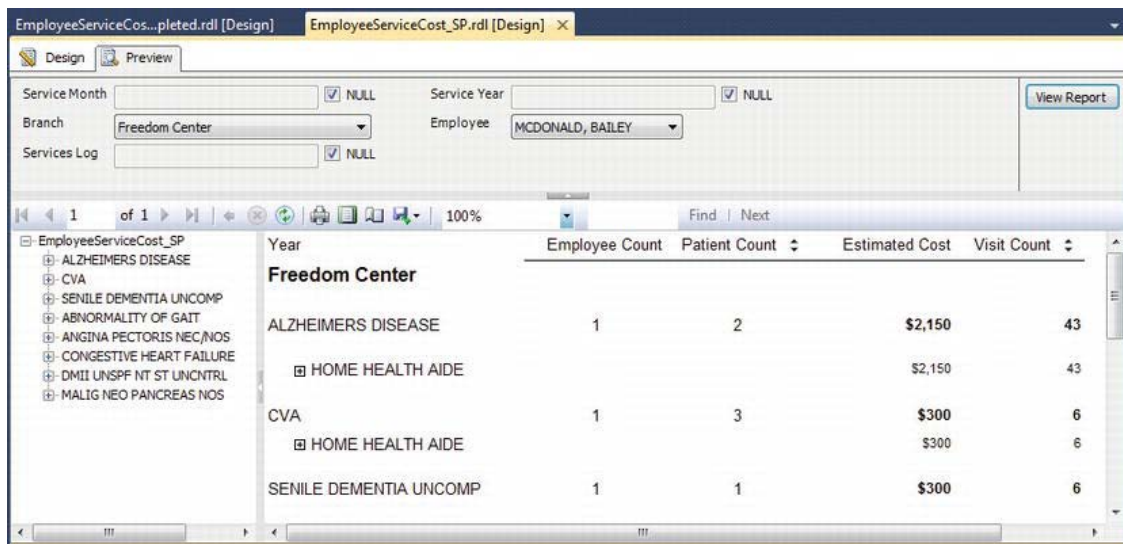


Figure 6-28 Report with populated parameter selections

You could perform the same steps for the ServiceLogCtgrID parameter and provide a valid drop-down list from the table values. However, since you may also be viewing the report in a custom report viewer that will also accept parameter values, this particular parameter value is of little use to you now for direct user input. That being the case, it will be beneficial to take advantage of another new and much needed feature, the ability to hide parameters. This functionality was added in Service Pack 2 for SSRS for SQL Server 2000 and is available all other releases since. Sometimes a parameter can and should be populated by events other than user input. In these instances, users will only be confused by seeing these additional parameters. In the Report Parameters dialog box, select the ServicesLogCtgrID property, and check the Hidden box. It will also be beneficial to modify the time-based parameters (Service Year and Service Month) for this report. Time-based values are often tricky to deal with because of the special formatting needs of the DateTime datatype, which can store years, months, and days as well as hours, minutes, and seconds. The procedures for setting up the Service Year and Service Month parameters with valid values is almost identical to the Branch and Employee procedures covered earlier, with the exception that the Service Year needs to default to the current year and not NULL.

The first step is to create a dataset for the Service Year and Month parameters based on the service date, which is the field ChargeServiceStartDate in the stored procedure. You will use the DatePart and DateName functions in the two queries to derive valid values. The valid values for the dates are contingent upon their existence in the table, so, for example, if your data contained values for 2009 and 2010, only those two years would show up in the drop-down list. Populating the date values in this way precludes the user from having to enter a date and also prevents the report designer from having to hard-code year and month values into the report. These Year and Month datasets have already been added to the report. However, we need to set the parameters Available Values to use Get values from query as we did previously. For ServiceMonth, use the Month dataset and set ServiceYear to use the Year dataset.

Listing 6-2 shows the two queries that drive the parameter values.

Listing 6-2. Parameter Value Queries

```
--Query to Derive Year
SELECT
    DISTINCT DATEPART(yy, ChargeServiceStartDate) AS Year
FROM
    Trx
UNION
SELECT Null AS Year ORDER BY Year

--Query to Derive Month
SELECT
    DISTINCT DATEPART(mm, ChargeServiceStartDate) AS DateNum
    , DATENAME(mm, ChargeServiceStartDate) AS Month
FROM
    trx
UNION
SELECT Null AS DateNum, Null AS Month ORDER BY DATEPART(mm, ChargeServiceStartDate)
```

To finish the report, add the Service Year field to the report by right clicking the BranchName field, Insert Row and then Inside Group – Below. Select the Year field from field selector list by placing the cursor over the top right corner of the blank space created underneath BranchName and clicking the field list box. Change the aggregation to be FIRST rather than the default SUM. Next, format it with a distinct color (in this case, dark salmon), set the alignment to be left justified and then resize the field to 12 points. Then, set the default value for all of the parameters with the exception of Service Year to be (NULL). To do this, use the Default Values tab under the Report Parameter Properties. Then select Specify values and click the Add button. Before you preview the report, it is important to set the default value for the year so that a valid Service Month selection is not based on the default Service Year field of NULL. This could potentially have undesired results; in other words, the user might select January and assume that it means January for the current year, when in fact it would be all occurrences of January.

To make the Service Year parameter default to the current year, go to the Parameters dialog box and set the Default Value option to the following expression:

```
=CINT(DATEPART("yyyy", Now()))
```

Also, in this report, we want to show the Service Year parameter before Service Month, so let's move the ServiceYear parameter above ServiceMonth parameter. Re-organizing the parameters like this changes their order at runtime as well. Open up the Report Data pane and expand the Parameters folder. Select the ServiceYear parameter and click the up arrow in the Report Data pane menu. You can preview the report and provide parameter values (see Figure 6-29).

■ **Note** Most of the data in the Pro_SSRS database is from 2009 and 2010. If the current year is defaulted to a different year, the data you see may not be the same as in Figure 6-29.

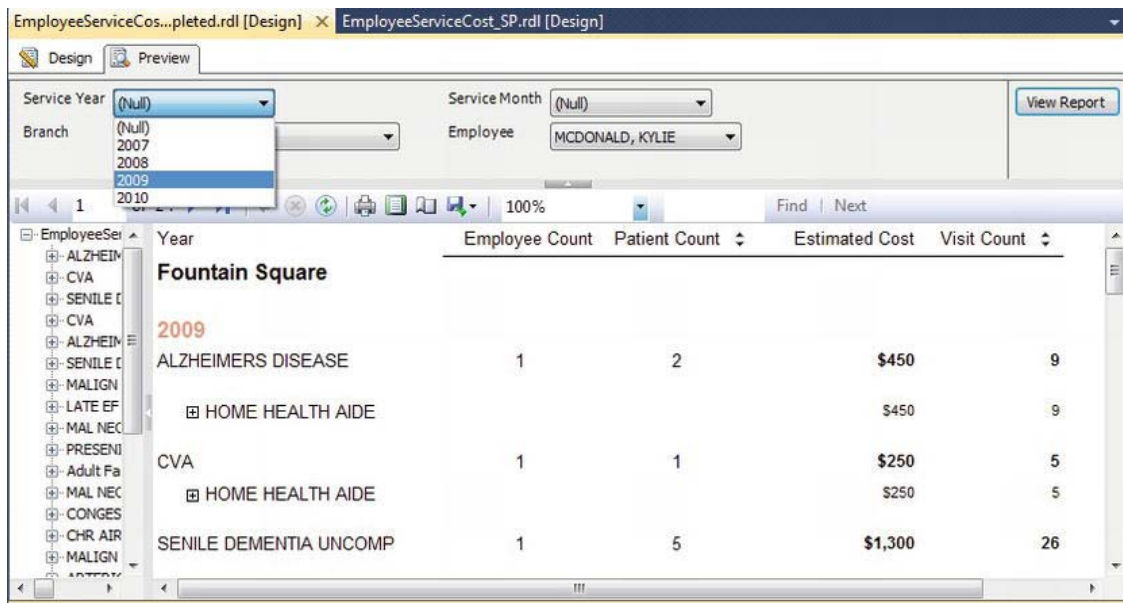


Figure 6-29. Report with valid year and month values

The EmployeeServiceCost_SP_Completed.rdl report in the Pro_SSRS project includes populated parameters.

Working with Multivalued Parameters

Multivalued parameters, an enhancement released for SQL Server 2005, represent probably one of the most awaited features for an SSRS update. Having the ability to select multiple values individually to feed into the report is a powerful feature that most other reporting applications take for granted, but was not available in SSRS for SQL Server 2000. Working with multivalued parameters to achieve the most usefulness from them, however, requires special design considerations, as mentioned in Chapter 3. The reason for this, especially when working with stored procedures, is that the multivalued parameters are passed back to the stored procedure as a string value. The only way to work effectively with multivalued parameters is to know that the query or stored procedure will evaluate all, one, or multiple values returned to it based on user selection. Because SQL Server does not evaluate a string in the same way it does a single value in a stored procedure, which honestly has been the bane of SQL developers for years, you have to go into multivalued parameters knowing that you may have to parse string values if you want to use stored procedures and multivalued parameters together with SSRS. For writers/logicians like us, this is a fun game. For others, who have to develop reports with multivalued input parameters for a large audience, this can be a nightmare. Rest assured that once you understand string manipulation techniques, multivalued parameters will be a worthwhile time investment.

SSRS does support multivalued parameters out of the box for queries as long as they are not stored procedures. To work with multivalued parameters with queries, all that is required is to use the IN keyword in the WHERE clause of the query that will be a dataset in the report. SSRS, in the case of multivalued parameters used in this way, will rewrite the query for you at runtime based on the

selections you choose in the report. This is a straightforward and well-documented process. However, we like using stored procedures for reasons already covered, and thus we have to make special accommodations, which you will see.

To accurately demonstrate how to use a store procedure with multivalued parameters, which we will affectionately refer to as MVPs henceforth, let's take a copy of the Employee Service Cost report with the assumption that you will redesign it to accept the Year and Month parameters as multivalues. To begin, you will have to first modify your base stored procedure. Previously, it was fine to evaluate the expression of your Year and Month parameters with the logic in Listing 6-3.

Listing 6-3. *Logic to Evaluate Year and Month Parameters Without MVPs*

```
1=Case
    When (@ServiceYear IS NULL) then 1
    When (@ServiceYear IS NOT NULL) AND @ServiceYear =
Cast(DatePart(YYYY, ChargeServiceStartDate) as int) then 1
else 0
End
AND
1=Case
    When (@ServiceMonth is NULL) then 1
    When (@ServiceMonth is NOT NULL) AND @ServiceMonth =
Cast(DatePart(MM, ChargeServiceStartDate) as int) then 1
else 0
END
```

However, now that you will be using MVPs, NULL values are not acceptable. The value of NULL in your logic was to select all values. This precluded you from accepting more than one value. For example, if you had the years 2007, 2008, 2009, and 2010 as valid values, you could either select all the values by selecting NULL or select only one value to filter the data. You could not have selected 2008 *and* 2009. With multivalued parameters you can. The only way to effectively use MVPs is through the WHERE clause of the query or stored procedure, with parameters, that feeds the report data. You will have to take advantage of the IN clause of T-SQL to make the best use of MVPs. Unfortunately though, it is not as simple as modifying the stored procedure to say Where value IN (@MyParameter), because SQL does not evaluate the IN clause as a string when using a stored procedure parameter. We can best explain this with the following example which can be seen by opening up the EmployeeServiceCost_MVP.rdl report in Pro_SSRS project from the book source code.

Let's say you make the Year and Month report parameters multivalued parameters. You can do this quite simply by checking the Allow Multiple Values box in the Report Parameter Properties window, as shown in Figure 6-30. Notice also that the Allow Null Value checkbox is unchecked. The Allow Null Value option cannot be checked if you want MVPs to work.

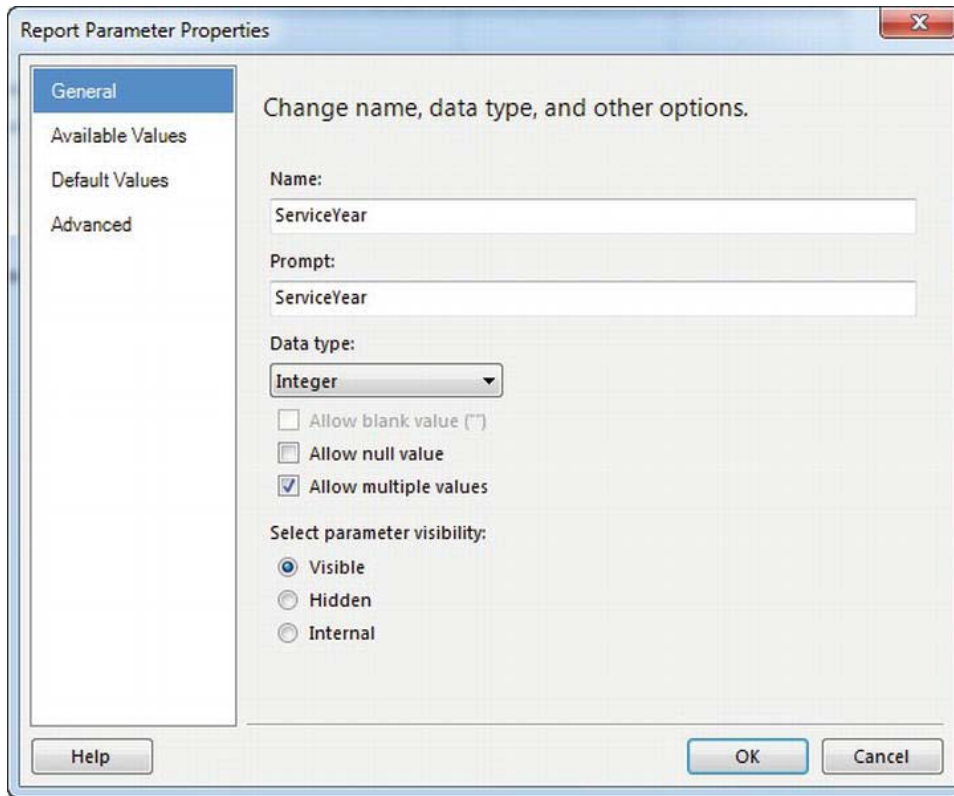


Figure 6-30. MVP options

If you were to execute the report now, you would see, as you did in the previous example using a dataset to populate the available values, that you are able to select one or more or all values for the year and month options, as shown in Figure 6-31.

Because the values for the MVP will be returned as a string—taking the year, for example, as “2007,2008”—this will not work with the stored procedure logic that you have defined. You will need to modify the stored procedure to use the IN clause so that the value will be equivalent to the following expression:

```
WHERE 1 = CASE WHEN CAST(DATEPART(YYYY, ChargeServiceStartDate) AS VARCHAR((20)) IN (@Year)
END
```

The problem here is that the variable @Year will be evaluated as a string and not an integer as it is defined in the stored procedure. If you were to select a single value—2007, for example—this would be fine because SQL would correctly evaluate the single value within the IN clause. However, when multiple values or Select All is chosen, SSRS passes a string such as “2007,2008,2009,2010”. When evaluated within the stored procedure, the query will fail. You need to first change the datatype of Year and Month to be a character or string value. So, you will choose varchar(20) for your stored procedure and parse out the values as they are passed in. Using varchar(20) will allow you to select a wide enough range to cover the Year and Month value strings.

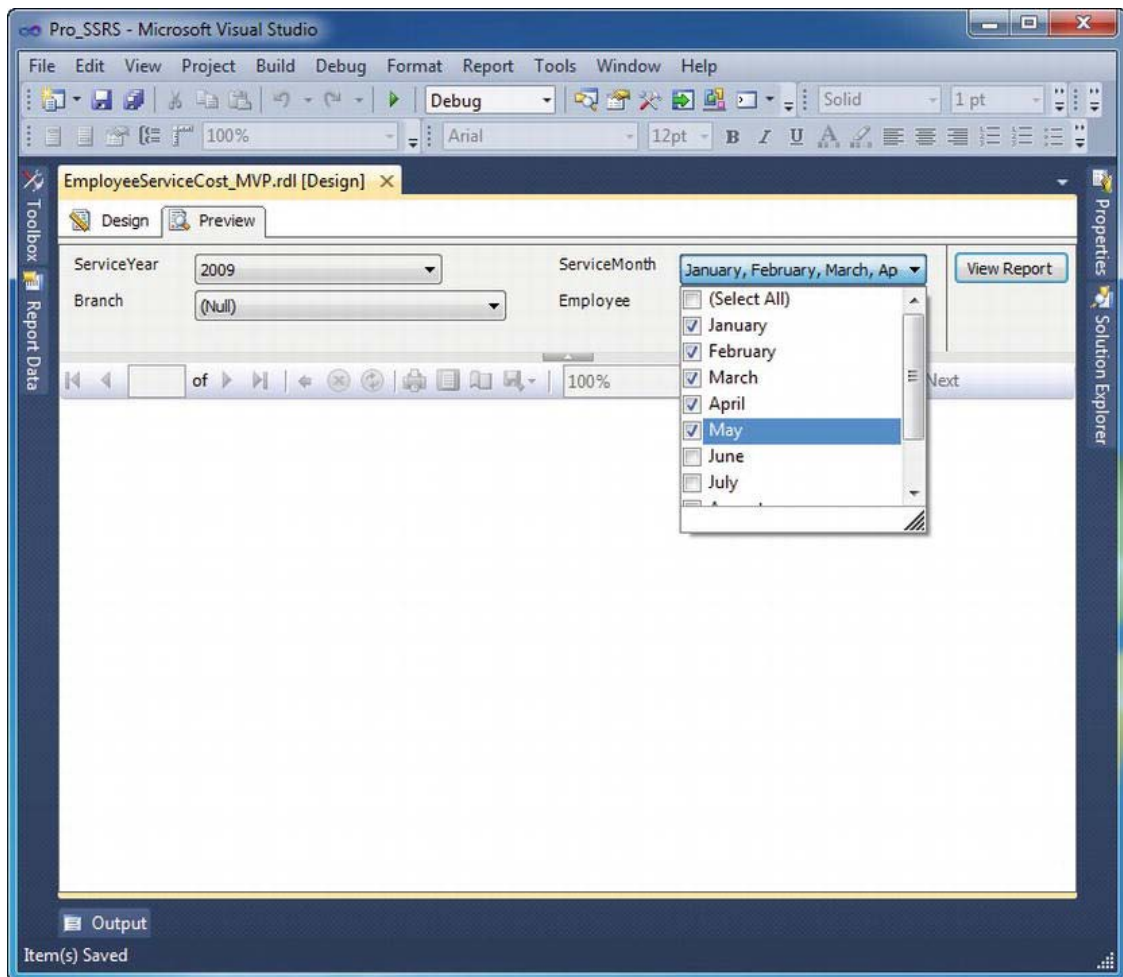


Figure 6-31. Multiple parameter selection

You also have to decide the best way to parse the string as it is returned from the report, both for performance and versatility. You have two effective methods for doing this, either dynamic SQL or a UDF. Creating dynamic SQL, which is essentially building a variable SQL expression using variables defined by user input, is cumbersome and syntactically challenging. Wrapping SQL statements within quotes and programmatically concatenating variables is time-consuming and often frustrating, and yields unpredictable results. What is worse is that it opens itself up to SQL injection hacks where users can interject values as strings that may execute statements that the developer did not intend. The best way to handle string values for MVPs is through a UDF that parses the individual values and feeds these into the IN clause of the query. Knowing that the values will always be returned in a comma-separated string makes loading the values into an accessible table much easier, by using a function designed for this purpose. This type of function is called a *table-valued function*, because the parsed rows of the input string are loaded into a table that can then be referenced as a subquery in the calling stored procedure.

Let's take a look at a parsing function that you will use in your stored procedure while working with MVPs. Listing 6-4 defines the UDF called `fn_MVParam`. This function is in the `Pro_SSRS` database that you have been using.

Listing 6-4. *fn_MVParam, String-Parsing Function*

```
CREATE FUNCTION dbo.fn_MVParam(@RepParam nvarchar(4000), @Delim char(1)= ',')
RETURNS @Values TABLE (Param nvarchar(4000))AS
BEGIN
    DECLARE @chrind INT
    DECLARE @Piece nvarchar(4000)
    SELECT @chrind = 1
    WHILE @chrind > 0
    BEGIN
        SELECT @chrind = CHARINDEX(@Delim,@RepParam)
        IF @chrind > 0
            SELECT @Piece = LEFT(@RepParam,@chrind - 1)
        ELSE
            SELECT @Piece = @RepParam
        INSERT @Values(Param) VALUES(@Piece)
        SELECT @RepParam = RIGHT(@RepParam,LEN(@RepParam) - @chrind)
        IF LEN(@RepParam) = 0 BREAK
    END
    RETURN
END
```

This function, when called from your `Emp_Svc_Cost_MVP` stored procedure, will return the parsed values from SSRS's multivalued parameter selection and allow you to use this as criteria for selecting data to include in the report. The key point of this function is that it uses several T-SQL functions itself, such as `CHARINDEX`, `LEN`, and `LEFT`, to populate the `@Values` table with the individual items from your report parameter string. The modification to the base `Emp_Svc_Cost` stored procedure shown in Listing 6-5 will be required to make the `Emp_Svc_Cost_MVP` stored procedure effectively work with the MVPs.

Listing 6-5. *Modification to WHERE Clause for MVP*

```
1 = CASE WHEN CAST(DATEPART(YYYY, ChargeServiceStartDate) AS VARCHAR(20)) IN
    (SELECT [PARAM] FROM fn_MVParam(@ServiceYear, ',')) THEN 1
    ELSE 0
    END
AND
1 = CASE WHEN CAST(DATEPART(MM, ChargeServiceStartDate) AS VARCHAR(20)) IN
    (SELECT [PARAM] FROM fn_MVParam(@ServiceMonth, ',')) THEN 1
    ELSE 0
    END
```

Notice that instead of saying `IN (@Year)`, for example, which will not work, you are calling your function `fn_MVParam`. The function takes two values: the string and the delimiter. In this case, you are using a comma as the delimiter.

When the report is run and the new function is called, you can see that you can select one, two, any combination, or all values from the populated drop-down, and you know that your stored procedure will effectively handle the parsing, evaluating, and criteria to deliver only the data that you want to see in the report, as shown in Figure 6-32.

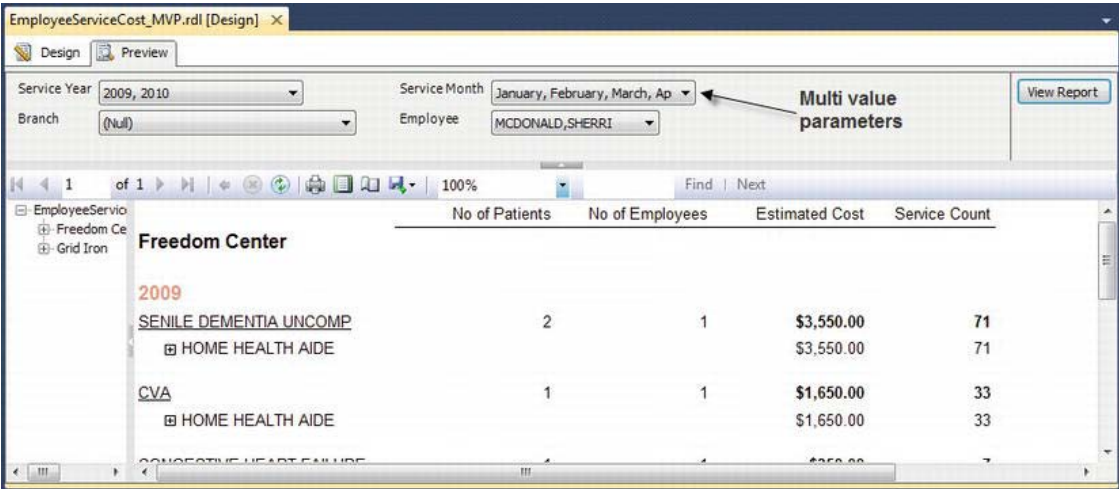


Figure 6-32. Report generated with multiple selection criteria

The completed report for multivalued parameters in the Pro_SSRS project is called EmployeeServiceCost_MVP.rdl.

Applying a Filter

You might recall from Chapter 2 that you enhanced the performance of your stored procedure, Emp_Svc_Cost, by removing the criteria that looked only at visits. You will now apply a filter to the report to take the place of the original query criteria so that only visits will be displayed.

You can use *filters* to exclude values from a report after the results have been returned by the query. Filters, in that sense, will prevent a re-query; however, the full dataset will be returned to the report. In the example in Chapter 2, you knew that a limited number of excess rows would be returned. You should use filters when a query parameter is not supported by the data provider or with report snapshots. You should also use filters in reports that address a specific request or solution, and that are based on the same stored procedure as other reports, because you can use filters without modifying an existing stored procedure. Here is a simple filter expression applied to the Table data region of your report that will exclude any rows that are not visits:

=Fields!ServiceTypeID.Value = "V"

To add the filter expression, on the Layout tab, right-click the upper-left section of the table, and select Tablix Properties. On the Filters tab, enter the previous expression so that it looks like Figure 6-33. Add the filter to the EmployeeServiceCost_MVP.rdl report included in the Pro_SSRS project from the book source code.

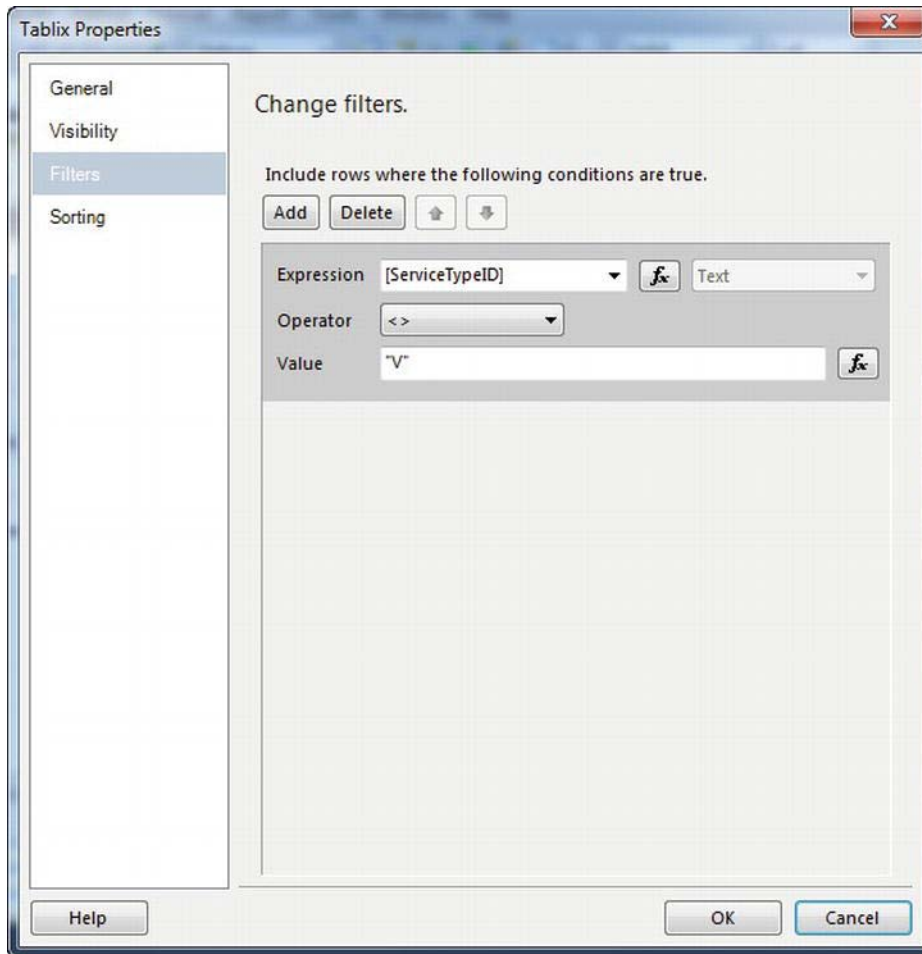


Figure 6-33. Filter dialog box to exclude non-visits

The completed report with a filter applied to a Tablix data region in the Pro_SSRS project is called EmployeeServiceCost_MVP_Filter.rdl.

Adding a Chart

SSRS provides a Chart data region that has a style similar to Microsoft Excel. Charts can be scoped within the current dataset or can use their own dataset. For this example, you will add a stacked bar chart to the beginning of the report that will show the top ten diagnoses and a count of the number of services for each diagnosis. This will essentially mirror the data provided in the report thus far. This report is now also grouped by Branch, which will automatically separate the values in the Branch group you have defined. You will want to emulate this for your chart. You have only three branches in this particular

dataset, so the result should be in line with the details of the report. Open the EmployeeServiceCost_MVP_Chart_Start.rdl report found in the Pro_SSRS project and follow these steps to add the chart to the report:

1. On the Design tab, click and drag the table you have already defined to make room for the chart.
2. Click and drag the Chart data region to an area above the table.
3. Right-click anywhere on the chart, select Change Chart Type, and in the Bar area select Stacked Bar.
4. Using the Chart_DS dataset already defined for your report; drag the DiagVisits to the Values area of the chart.
5. Drag the Diagnosis field to the Category Groups area of the chart.
6. Drag the Patient_Count field to the Values area underneath the DiagVisits series of the chart.
7. Resize the chart so that it aligns with the table below it. You can select both report elements, and on the toolbar select the Make Same Width icon.
8. Right click on the Diagnosis Category Group and then click on the Filters tab. Click the Add button to add a filter to the category group. Because you want to make the report show only the top ten diagnoses, you need to add a filter to this grouping. You will use the Top N operator to make this happen, as shown in Figure 6-34, based on the top ten diagnoses by a sum of the Diagnosis visit count. While you're in the Diagnosis Category Group properties, click on the Sorting tab. Change the default sort by to the expression =SUM(Fields!DiagVisits.Value) and select A to Z for the sort order.

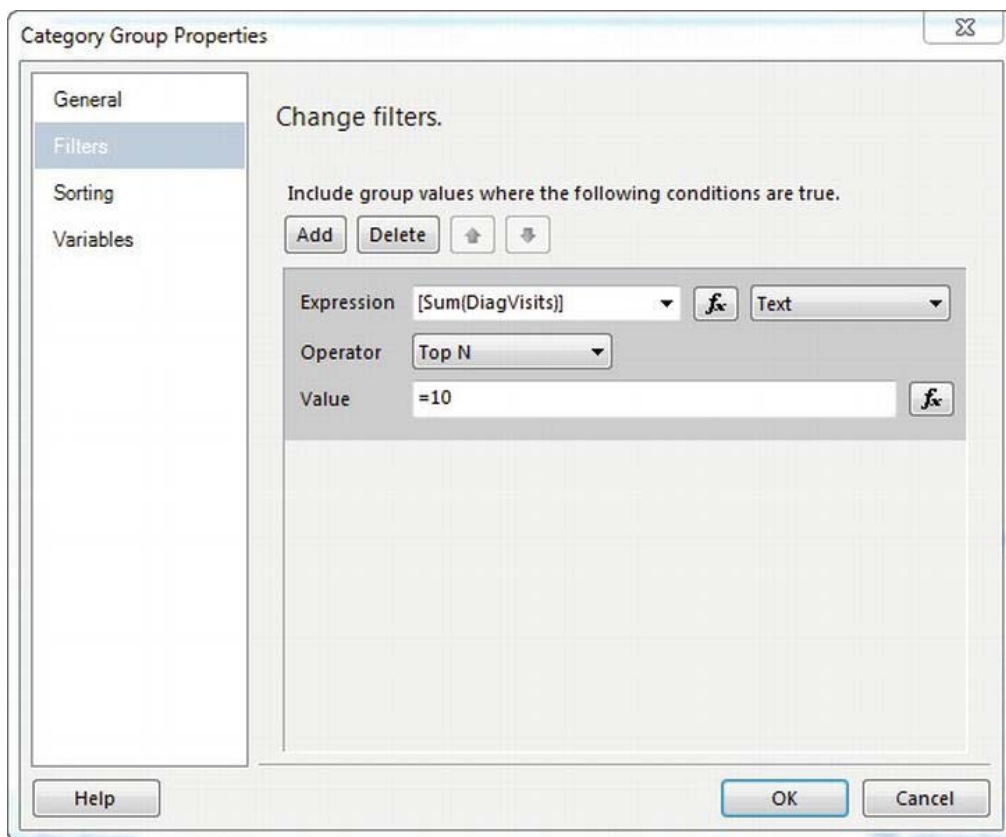


Figure 6-34. Filter value for top ten diagnoses

Finally, you can preview the report. Sometimes a report needs a chart up front for a quick view of the data prior to analyzing the details. Someone perusing this report might find it interesting that the chart shows, for example, that the Physical Therapy NEC diagnosis seems to be more prolific in the Nested Valley branch. This initial preview might warrant more investigation, which can be gleaned from the details in the report.

When previewed, the report should look like Figure 6-35.

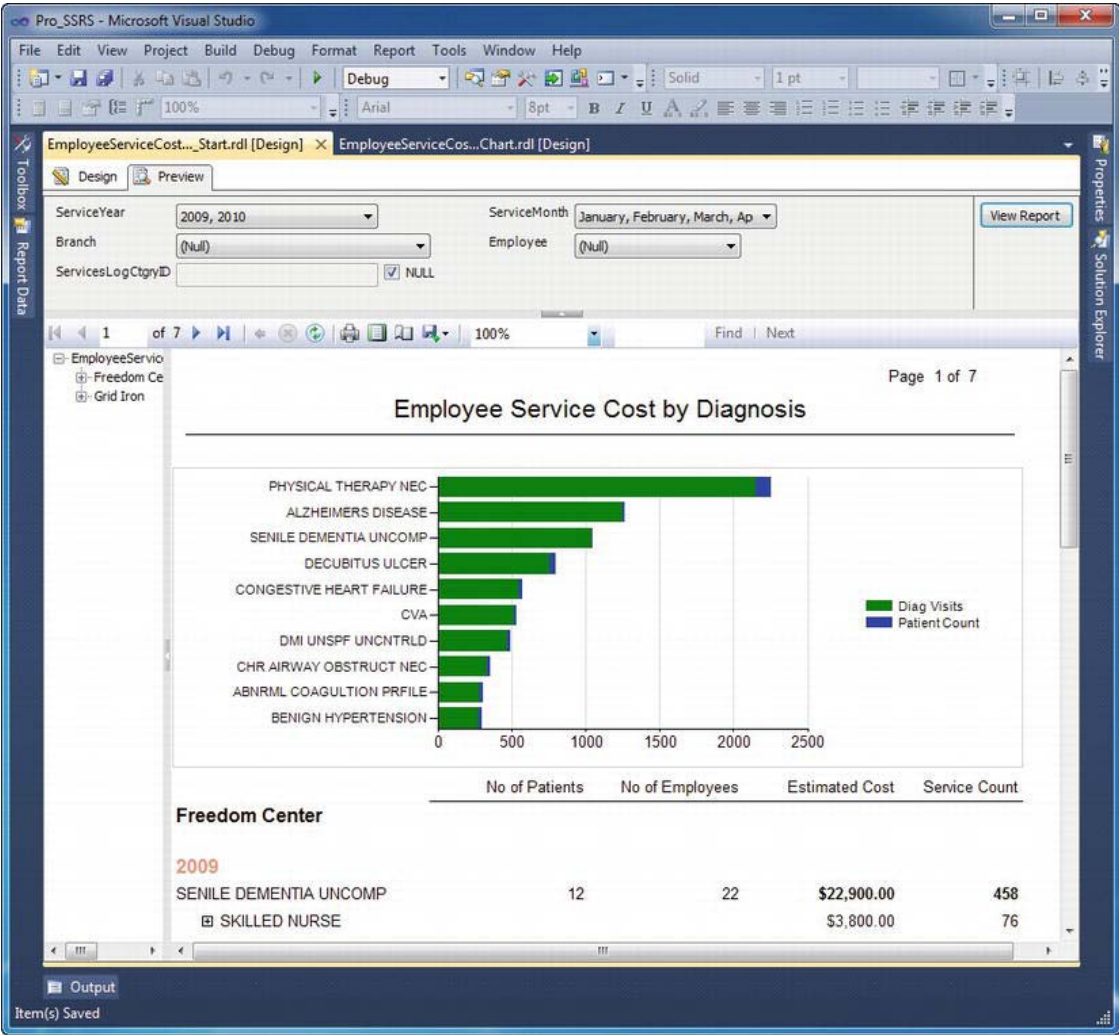


Figure 6-35. Employee Service Cost report with a chart

The Chart data region has many properties that you can apply, as covered in Chapter 5; however, the appearance of the stacked bar is suitable for your report and can be deployed as is. Nevertheless, with a few slight tweaks like removing the chart title, axis titles, and choosing the Default color scheme, the final report starts to take on a cleaner look and feel. The completed report with a filter applied to a Chart data region in the Pro_SSRS project is called EmployeeServiceCost_MVP_Chart.rdl.

Adding Tablix Elements

SSRS 2008 introduced a significant change in report design by way of the Tablix data region. Essentially, the Tablix, as covered in detail in Chapter 4, combines aspects of the Table and Matrix data regions. The Table region available in SSRS 2005 worked well with row data of variable length based on the dataset. In turn, the Matrix data region included support for a variable number of columns. Reporting Services 2008 allowed use of either control to include custom row or column groupings anywhere in the report. As you will see in the subsequent addition to the Employee Service Cost report, adding column groupings to a table data region is straightforward.

In the section “Configuring Report and Group Variables” that follows, you will add a column grouping to the `EmployeeServiceCost_Tablix_Start` report using the Year field. The Year field, as you may recall, indicates what year the types of services were rendered for the patients. So far in the report, you have a distinct patient and employee count as well as estimated cost and visit count. So, what if you wanted to see each of these values grouped by the year of service for multiple years?

You will perform the following actions to add a column group for the Year to the Employee Service Cost report:

1. Right-click the `[CountDistinct(PatID)]` cell in the table and select **Add Group** and then **Parent Group** under the **Column Group** section.
2. In the **Group by Expression** field, select `[Year]`. Do not check the **Add Group Header** or **Add Group Footer** button. Click **OK**.
3. Since we want to have groups that are independent of one another, we now need to add what is called an **Adjacent Group**. On the new `[Year]` field that we just created (which should be just above the **No of Patients** label), right-click and select **Add Group** and then **Adjacent Right**. Group on the `[Year]`, as you did in step 2. Click **OK**. This new column will hold all of the values in the **No of Employees** column that currently exists. In step 5, we will shift all of the values to be underneath the new `[Year]` columns.
4. Perform step 3 two more times, each time using the right-most `[Year]` field.
5. Now that we have three columns that just have the `[Year]` in the column groupings, we need to move our existing values underneath of them. At this point, the design surface should look like Figure 6-36. Next select all of the fields from the **No of Employees**, **Estimated Cost**, and **Service Count** columns. Be sure to select the header text too! Then cut and paste the values from their original cells into the three empty columns under our new **Adjacent Groups** that we created in steps 2 and 3.
6. The three columns on the right are no longer needed. Delete them by right-clicking the column and choosing **Delete Columns**. You can select all three of them by holding the **CTRL** key as you select them.
7. Finally, adjust the widths of the columns with enough room to show the labels as shown in Figure 6-37.

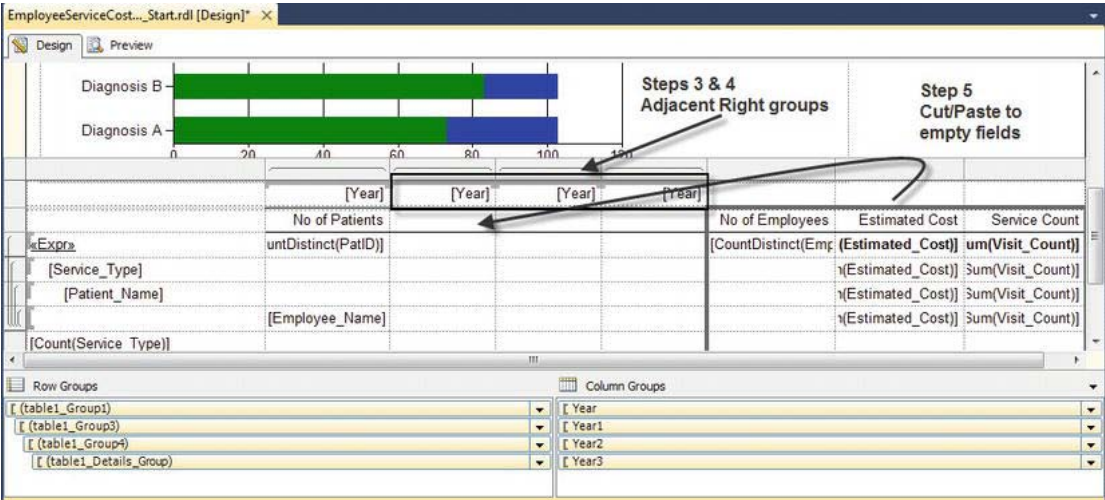


Figure 6-36. Design layout for before moving fields over

When you are finished, the report should resemble Figure 6-37 on the Design tab. You can also see the visual cues of the groupings, both row and column on the table as well as the Row Groups and Column Groups regions, of which you now have four groups for each. The completed report with Column Groups applied in the Tablix data region in the Pro_SSRS project is called EmployeeServiceCost_Tablix.rdl.

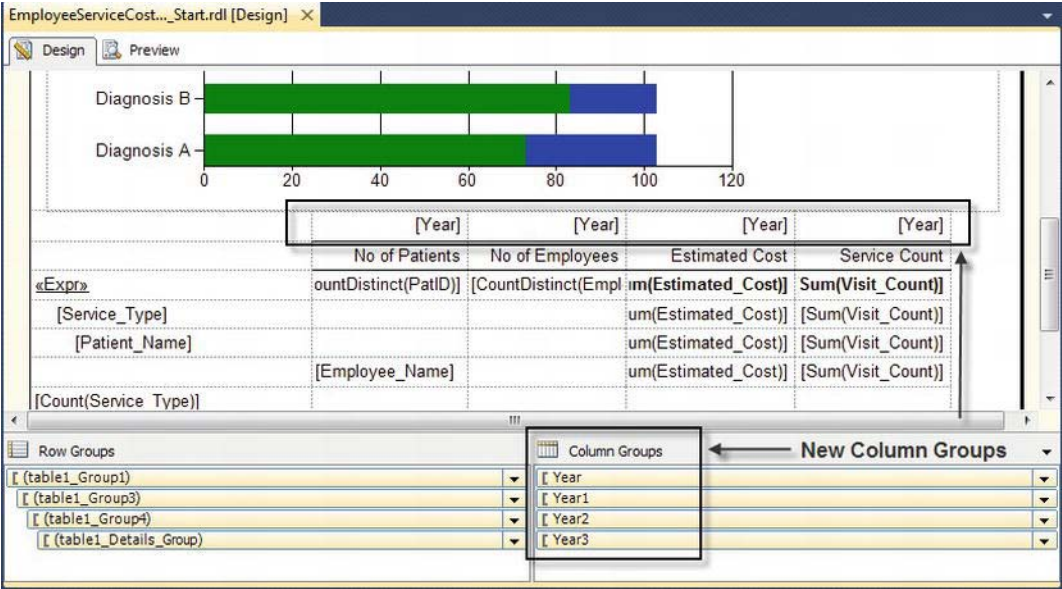


Figure 6-37. Design layout for column groupings

When you preview the report and select multiple years, as in Figure 6-38 where years 2009 and 2010 and all months are selected, you can see the column groupings automatically expand horizontally and group the values accordingly. For example, there are 75 Home Health Aide visits in 2009 for Congestive Heart Failure and only 16 visits in 2010 for the months selected.

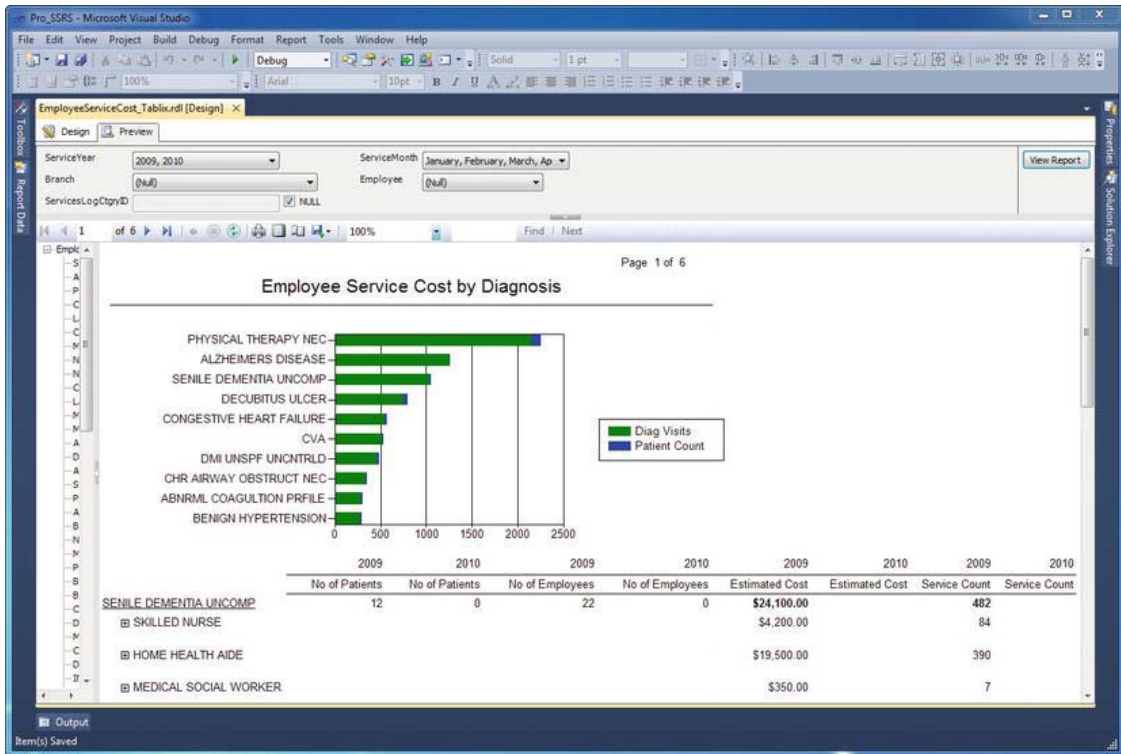


Figure 6-38. Preview of Employee Service Cost report with column groupings

Configuring Report and Group Variables

SSRS has implemented global variables since its initial release in SQL 2000. These variables, such as `ExecutionTime` and `UserID`, can be used in reports. For example, the expression `=Globals!ExecutionTime` returns the time at which the report was executed. This value, unlike that returned by the expression `Now()`, does not change after the report is rendered, so paging through the report will not change the value of the global variable.

SSRS 2008 introduced two new types of variables—report variables and group variables—that can be configured at design or runtime to be static throughout report execution and viewing. You will use a report variable in this section to create a threshold for the `Visit_Count` field so that when the report is run, you can calculate whether the number of visits have met the report variable threshold. You will use the same variable again in the new SSRS Gauge control.

To configure a report variable, go into design mode, click **Report Properties** under the **Report** menu, and then select the **Variables** tab. You will need to add the report variable as in Figure 6-39. The

threshold that you will set will be for the average number of visits per day. The method to derive this value is a simple calculation of a distinct count of the unique field Trx.ServicesTblID divided by a range of days, say 365 for a year. Our calculation for the Pro_SSRS database reveals an average of 44 daily visits for 2009. The next step, then, is to create a report variable that will hold this threshold value.

To set up a report variable, open up the EmployeeServiceCost_Variables_Start.rdl report from the Pro_SSRS project and go to the Design tab, click Report Properties under the Report menu, and click the Variables tab. Click Add and enter **Threshold** as the name of the variable, **44** as the value, and remove the check from Read-Only, as shown in Figure 6-39.

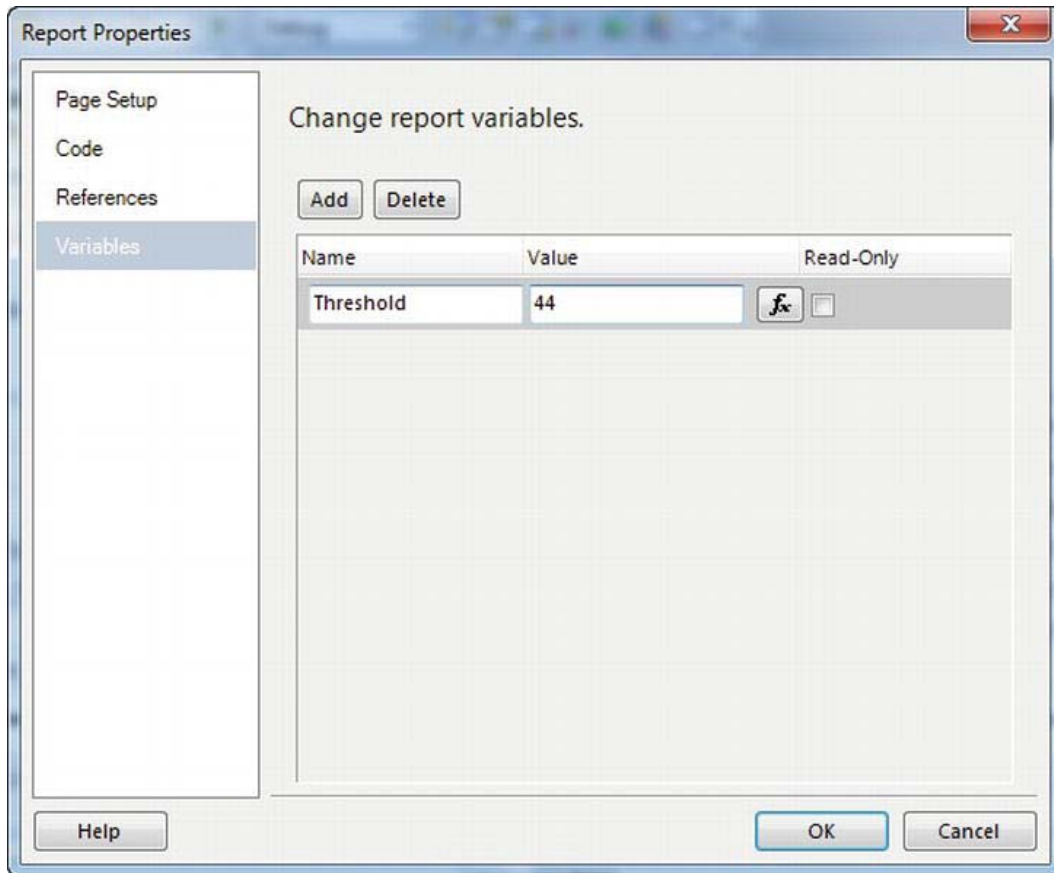


Figure 6-39. Adding a report variable

Like report variables, group variables can be configured for one or more row or column groups. They are useful for storing static values, such as differing tax values for products and product subcategories, or subgroups.

Adding the Gauge Control

Gauge controls are one of the most aesthetically appealing types and are useful for reports that focus on Key Performance Indicators (KPIs). Such reports are designed to be viewed in dashboard format for at-a-glance views of data, as you saw in Chapter 5. In this section, we will combine the report variable we created in the previous section with one of these new controls to create a visual cue within the Employee Service Cost report. This report will show us how many visits have been made per day for the time period we are analyzing.

You can determine the average number of visits by a very simple calculation dividing the sum of the visit counts by a distinct count of the `ChargeServiceStartDate` field. You can perform that computation using the `Emp_Svc_Cost_MVP` stored procedure. This average number of visits value will become one of the two pointers in the gauge control, which will be the multiple bar gauge. The other pointer in the control will hold the threshold value of 44.

To begin, drag a Gauge control from the Toolbox onto the report to the right of the chart. Just make sure it is inside the Rectangle. Select the Multiple Bar Pointers gauge. There will be three bars by default. You will only use two of the bars for this report, `LinearPointer1` and `LinearPointer2`. Go ahead and delete the small one (`LinearPointer3`) by right-clicking it and choosing to `Delete Pointer`. Assign the expression `=Variables!Threshold.Value` to `LinearPointer1` by right-clicking the pointer and selecting `Pointer Properties`. In the Properties window, assign the variable expression value as shown in Figure 6-40.

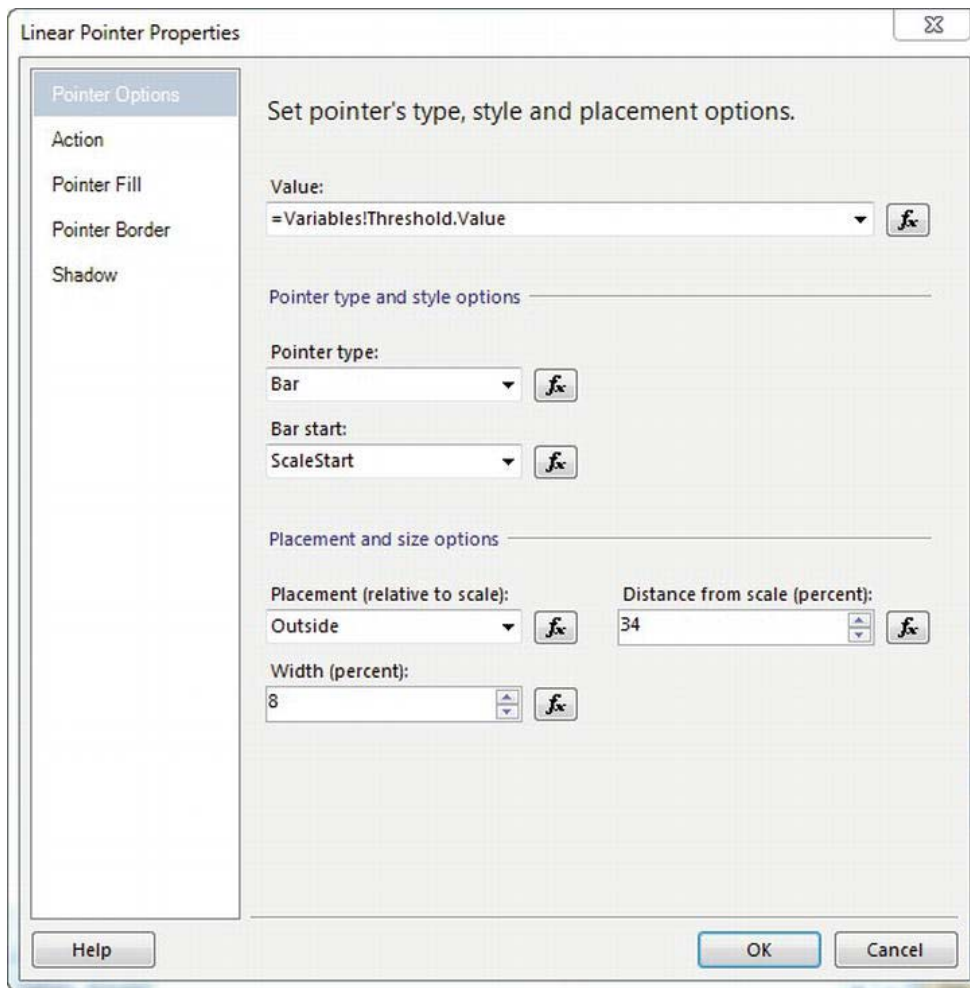


Figure 6-40. Assigning a report variable value to the gauge pointer

Next, assign the expression to determine the average number of visits for the time frame selected from the parameters to LinearPointer2. Because the multivalued parameters are query parameters, you know that the data in the report will be limited to the values selected from the Month and Year parameters. This would not be the case if you were using report parameters and subsequently filtering the data within the report. Knowing, for example, that the number of days over which to calculate the average will change based on the Month and Year parameter values selected makes it easy to do the calculation for LinearPointer2. You can write that calculation as

```
=SUM(Fields!Visit_Count.Value) / CountDistinct(Fields!ChargeServiceStartDate.Value)
```

Your final step is to set the color coding for the pointer such that the pointer will show red for 44 and under, and green for over 44. Right-click on LinearPointer2 and select the Pointer Properties option.

Next, click the Pointer Fill tab. Click the Expression button to the right of the secondary color, enter the following expression, and click OK:

```
=IIF(Sum(Fields!Visit_Count.Value) / CountDistinct(DAY(Fields!ChargeServiceStartDate.Value) & YEAR(Fields!ChargeServiceStartDate.Value)) < Variables!Threshold.Value, "Red", "Green")
```

The final view of the report in Figure 6-41 has the parameter value for ServiceYear of 2009 and 2010 selected. You can see that the threshold has been met for the year and month values selected. The completed report with all the final touches can be seen in the Pro_SSRS project is called EmployeeServiceCost_Variables.rdl.

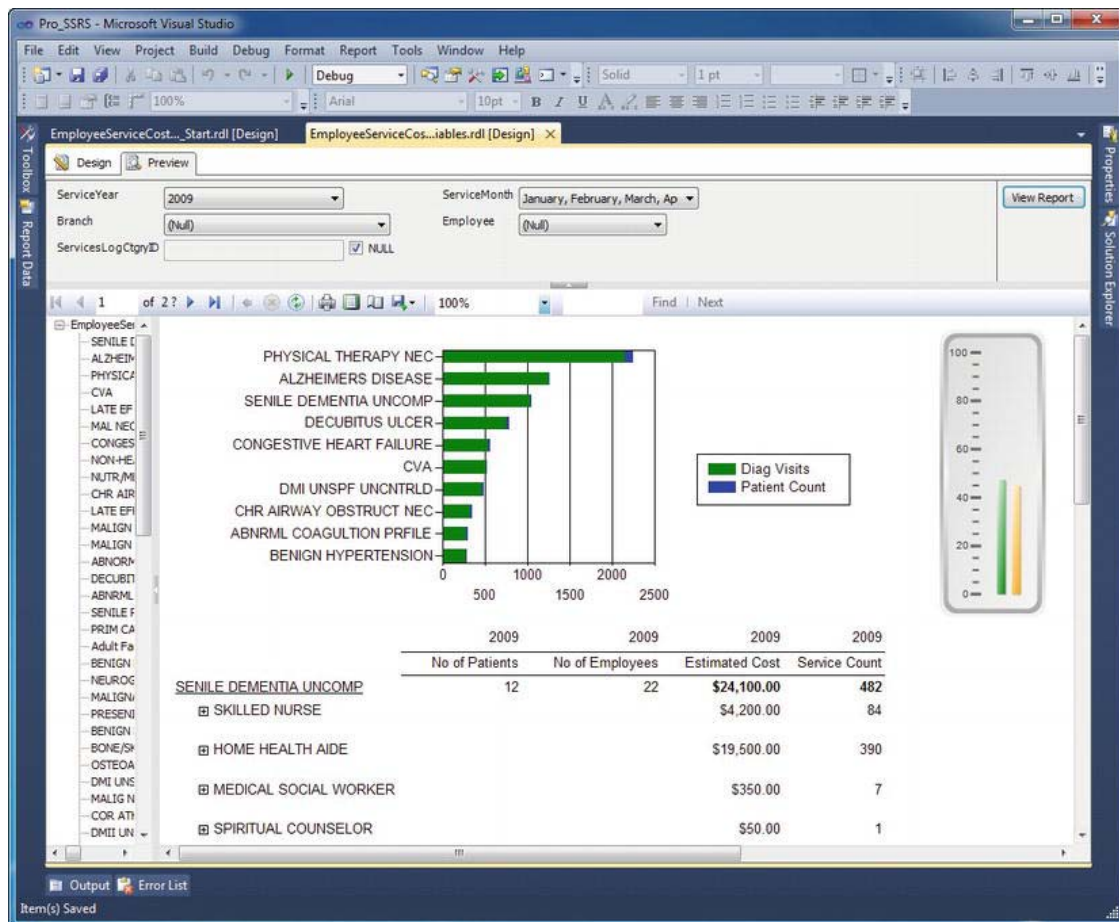


Figure 6-41. Final view of the report with the gauge control

Adding the Final Touches

In many projects, the final touches can be quite time-consuming. You now have the Employee Service Cost report to the point where it is functioning the way you expect and has had formatting applied to some extent during development. You have only a few final elements to apply before you call the report complete and before it's ready to deploy to your SSRS Web server for production:

- Adding a page header and footer
- Adding a report title
- Adding page numbers
- Adding the report execution time

To add a page header and footer to the report, select the Report menu and then Add Page Header and Add Page Footer. This will add the two new sections to the report where you will add the values that will print on every page.

In the Toolbox, you'll see two report items, a textbox and a line, that you can use in the page header and footer sections. First drag two textboxes to the header and one to the footer. Then align one of the header textboxes to be the same width as the table, enter your report title as Employee Service Cost by Diagnosis, change the font size to 16 points, and apply bold formatting. Next, drag a line into the header section, and position it between the chart and the report title textbox you just created.

In the second textbox in the header section, add the following expression based on global parameters as defined in the Edit Expression window:

```
= "Page " & Globals!PageNumber & " of " & Globals!TotalPages
```

Finally, in the textbox in the footer, add the following expression for the report execution time:

```
=Globals!ExecutionTime
```

You are now ready to preview your report one last time before you deploy it to your users. This time, let's take a look at the final version in the browser (see Figure 6-42). This is what the report will look like when it has been deployed to the Web server. (We will discuss methods for deploying reports in Chapter 8). After previewing the report, scroll down to the bottom of the report to see the report execution time in the footer. The completed report with all the final touches can be seen in the Pro_SSRS project is called EmployeeServiceCost_Final.rdl.

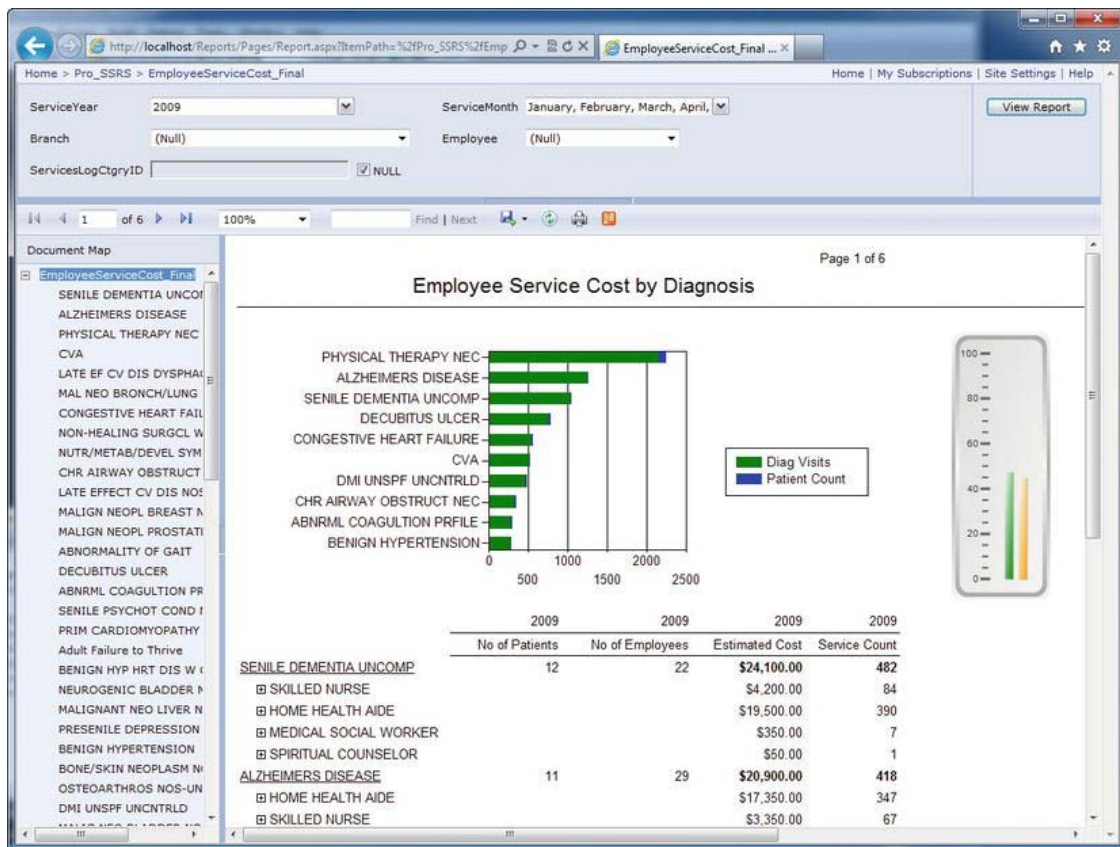


Figure 6-42. Report rendered in the browser

Summary

It seems as if we have covered much ground in the actual design of a reporting solution with SSRS. However, at the same time, we have only scratched the surface of getting to the raw power and flexibility of SSRS. We have yet to show how to interweave custom assemblies to perform specific functions that go beyond basic expressions. You will also be working with other data regions in other parts of the book that we have not touched on here. Additionally, you have been working with only a small number of reports in this chapter; often in a business, especially when facing migrating existing reports to SSRS, you will be working with many reports simultaneously. Deploying, administering, and securing these reports are going to become critical next steps.

Luckily, a robust and flexible design environment is only one component of SSRS. In the upcoming chapters, you will deploy, secure, and analyze the performance of the reports you are designing here, using a variety of methods.

Using Custom .NET Code with Reports

SSRS offers software developers a variety of options when it comes to customizing reports through code. These options give software developers the ability to write custom functions using .NET code that can interact with report fields, parameters, and filters in much the same way as any of the built-in functions. To give just two examples, you can create a custom function that does the following:

Implements a business rule and returns true or false based on the logic. You can use such a function as part of an expression to change the value or style of a field based on the fields or parameters passed to the function.

Reads data from sources not otherwise available to SSRS 2012 directly. You can do this by having your custom code read data directly from the source. In this chapter, you will examine how to read data from an XML file. The sample code for this chapter also includes an example of reading data from a web service.

In short, using custom .NET code gives developers the ability to extend the capabilities of SSRS far beyond those that are available out of the box. This chapter will cover the following:

Custom code for use within your report using code embedded in the report. This method is the simplest way to add custom code to your report, and it deploys along with your report since it is contained in the RDL. However, it limits what you can do since you can only use a small subset of the .NET libraries, it must be written in VB .NET, and it offers limited debugging support. You are also limiting the reusability of the code since you must manually move and make changes to any code that might be included in different reports.

Custom code for use within your report using a custom assembly called by the report. This method is more involved to implement and more difficult to deploy, but it offers you nearly unlimited flexibility. Your custom code has the full power of the .NET framework at its disposal and has the added benefit that you can use the custom code across multiple reports using a single assembly. You can also use the full debugging capabilities of Visual Studio while developing your custom assembly, as well as include it in any code repositories such as Team Foundation Server.

Generally, you will add custom code to your report when you need to perform complex functions and need the capabilities of a full programming language to accomplish them. However, before you embark on writing custom .NET code, you should first evaluate whether using the built-in expression functionality can meet your needs. Using .NET embedded code or custom assemblies instead of native SSRS functions can give you serious performance degradation. If you don't need to use .NET to perform

a task in an SSRS report, you probably shouldn't take on the added overhead of in report code or a custom assembly.

Using Embedded Code in Your Report

Using embedded code is by far the easiest way to implement custom .NET code in your reports, for two main reasons. First, you simply add the code directly to the report using the Report Designer's user interface (UI) in either BIDS or Visual Studio. Second, this code becomes a segment within the report's RDL file, making its deployment simple because it is a part of your report and will be deployed with it in a single step.

Although embedded code is easier to use, there are a few considerations that you should take into account:

Embedded code must be written in VB .NET. If you are a C# programmer or use some other .NET-compatible language as your primary development language, this may force you to use the custom assembly for all but the simplest of functions. Chances are that if you are developing complicated code, you will want to go with the custom assembly route anyway for the debugging and source control options.

All methods must be instance based. This means the methods will belong to an instantiated instance of the code object and you cannot have static members.

Only basic operations are available. This is because, by default, code access security will prevent your embedded code from calling external assemblies and protected resources. You could change this through SSRS security policies, but it would require granting FullTrust to the report expression host, which would grant full access to the CLR and is definitely not recommended. If you need these capabilities, use custom assemblies so you can implement security policies to grant each assembly only the security it needs. You will look at custom assemblies and how to set security for them in the "Deploying a Custom Assembly" section.

Before you run the examples for this chapter, make sure to read the `ReadMe.htm` file included with the sample code, available for download from the Source/Download page at www.apress.com. It is located in a file in this chapter's samples root folder. If you have the code open in Visual Studio 2010, it will be under the Solution Items folder. It contains setup and configuration steps that are required before running the examples.

Let's take a look at how this feature of SSRS works by adding some embedded code to one of the reports you have already created. In this case, start with the sample Employee Service Cost report included with this chapter. It is a slightly modified version of the Employee Service Cost report you created in Chapter 6. We will show you how to use the embedded code feature to add a function that will determine whether you have exceeded a certain cost per visit for a type of treatment in a given time period. You will then use that function to determine the color of one of the text fields in the report to help draw attention to those specific treatment types.

■ **Note** In this chapter's example, we'll use a slightly modified version of the report created in Chapter 6 so that the employee report parameter will include a treatment with enough patients who have exceeded the maximum number of visits and average cost.

Using the ExceedMaxCosts Function

Listing 7-1 is the full listing of the custom code that you will add to the Employee Service Cost report. It is a simple function, called `ExceedMaxCost`, which determines whether a certain treatment type has exceeded an average cost over some period of time. This allows you to identify cases for review to determine why they have such a high cost of services.

Listing 7-1. The ExceedMaxCost Function

```
Function ExceedMaxCost(ByVal visitCount As Integer, ByVal estCost As Integer) As Boolean
    ' Our businesses logic dictates that we need to know whether
    ' the average cost per visit exceeds a certain value but only when the visit
    ' count is greater than 10

    If ( visitCount > 10)
        If (estCost/visitCount > 45)
            Return True
        End If
    End If

    Return False
End Function
```

If you are following along with the code in the book, you will need to create a new Visual Studio 2010 BI project, selecting the Reporting Services subcategory and Report Server Project type, as shown in Figure 7-1. For this example, call the solution Chapter 7 and the project Reports.

■ **Note** BIDS is now built directly into Visual Studio with the release of SQL Server 2012. Whether you are creating this from BIDS or from Visual Studio, the resulting project and solution will be the same.

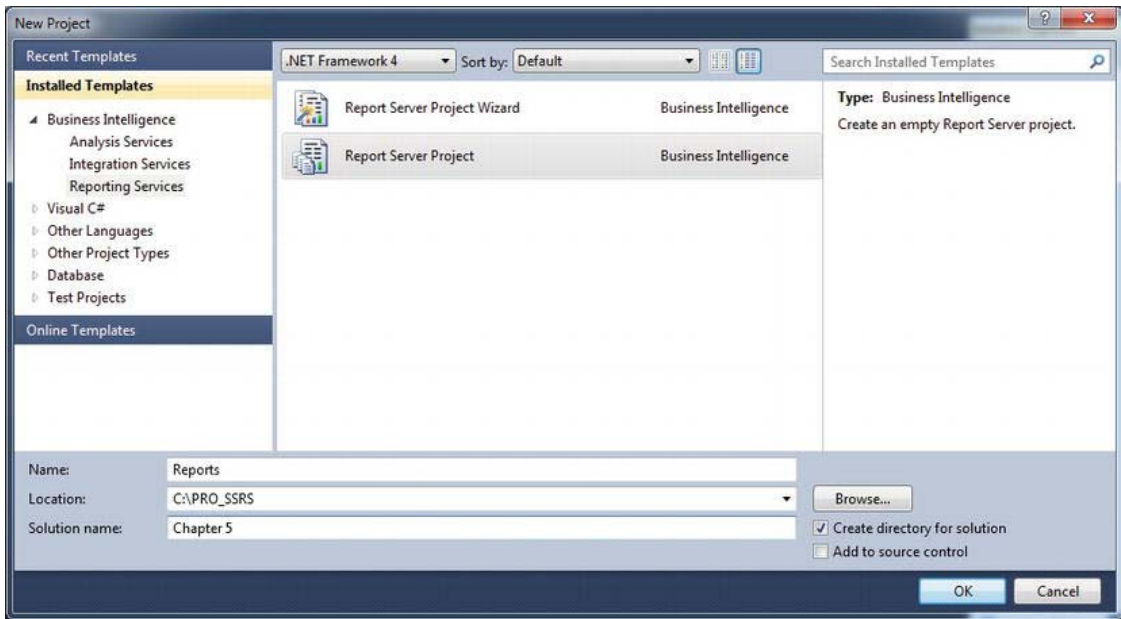


Figure 7-1. Creating a Visual Studio BI project

To add the existing `EmployeeServiceCost-NoCode.rdl` file to your new project, right-click the `Reports` folder, and select `Add ► Existing Item`, as shown in Figure 7-2. Alternatively, with the `Reports` project highlighted, select `Add ► Existing Item` from the menu. Next, browse to the location where you installed the Chapter 7 samples, and select `EmployeeServiceCost-NoCode.rdl` from the `Reports` folder. You will also need to add the shared data source by adding an existing item and picking the `Pro_SSRS.rds` file from the same folder where the report was located.

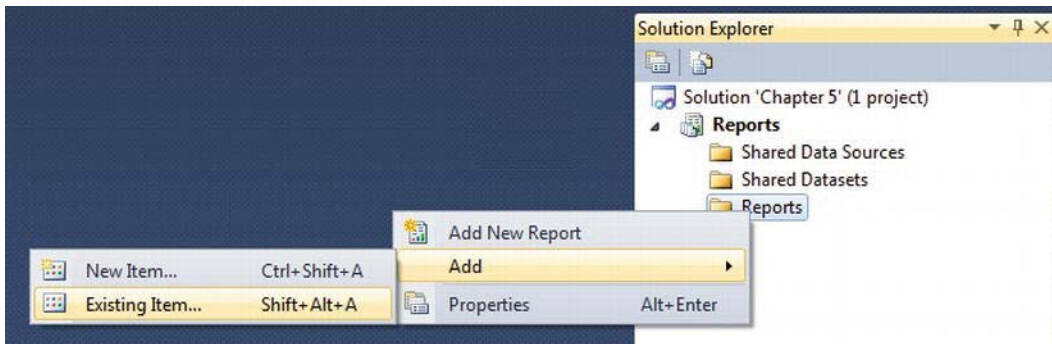


Figure 7-2. Adding `EmployeeServiceCost-NoCode.rdl` to your project

To add the code from Listing 7-1 to the `Employee Service Cost` report, first open the report by double-clicking it or by right-clicking it and selecting `Open` in the `Solution Explorer`. Next, with the report on the `Design` tab, select `Report Properties` from the `Visual Studio Report` menu; alternatively,

right-click within the report design area, and select Properties. On the Report Properties dialog box's Code tab, add the code from Listing 7-1 to the Custom Code box, as shown in Figure 7-3.

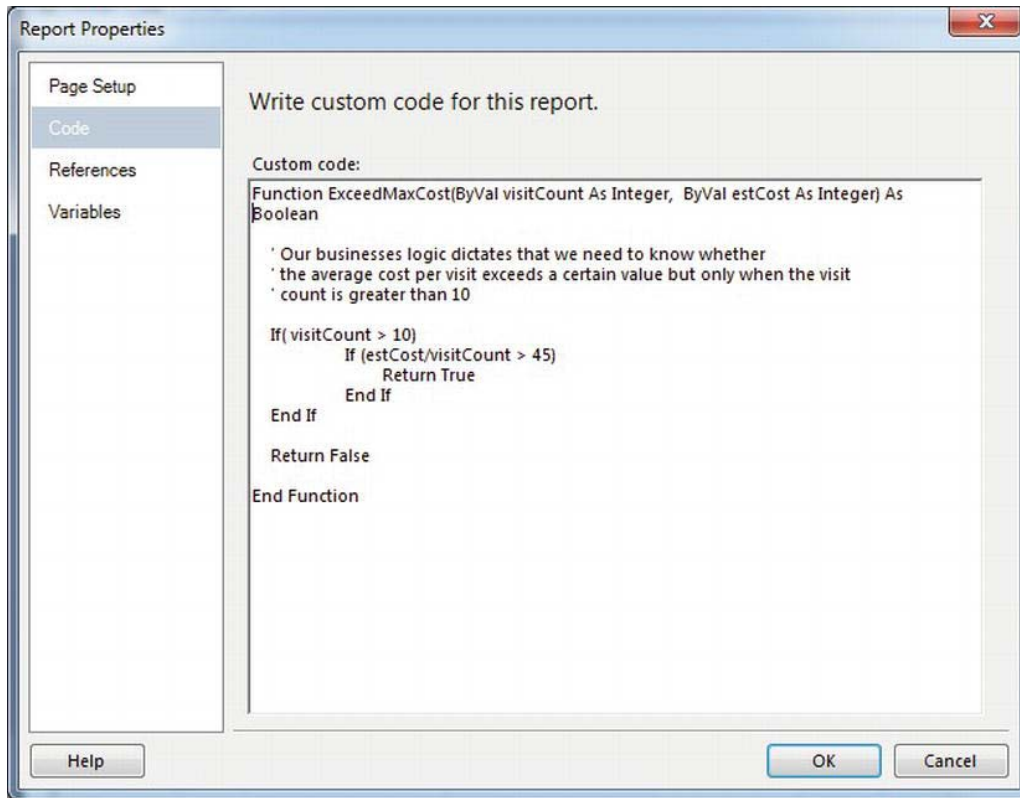


Figure 7-3. Entering embedded code in the custom code editor

■ **Note** You must enter the function declaration (the first line) as a single line in the embedded code editor, or you will receive an error when you try to preview the report. It is shown with returns in Listing 7-1, but should be entered into the embedded code editor without them.

Now that you have defined your custom code, you'll want to use it to highlight the treatment types that have exceeded the maximum visit count. To do this, you need to access the `ExceedMaxCost` method as part of an expression.

Methods in embedded code are available through a globally defined Code member. When a report's RDL file is compiled into a .NET assembly (at publish time), SSRS creates a global member of the class called `Code` that you can access in any expression by referring to the `Code` member and method name, such as `Code.ExceedMaxCost`.

Listing 7-2 shows how to use a conditional expression in the Color property of a textbox to set the color of the text depending on the return value of the function call.

Listing 7-2. Using a Conditional Expression

```
=IIF(Code.ExceedMaxCost(Sum(Fields!Visit_Count.Value,
"table1_Group1"),Sum(Fields!Estimated_Cost.Value, "table1_Group1")), "Red", "Black")
```

The method ExceedMaxCost determines whether the type of treatment has had more than 10 instances in the time span and the average cost was more than 45 per visit, and returns True if so or False if not. Using a Boolean return value makes it easy to use the method in formatting expressions, because the return value can be tested directly instead of needing to be compared to another value.

Using the IIF native SSRS function, we can pass the custom ExceedMaxCost directly and evaluate the return value. IIF evaluates the first parameter and will return the second parameter when True and the third when False.

When a treatment type exceeds the maximum average cost allowed, ExceedMaxCost returns True, which sets the value of the textbox Color property to Red, which in turn will cause the report to display the text in red. If not, then ExceedMaxVisits returns False, and the Color property is set to Black.

Using the ExceedMaxCost Function in a Report

Now we'll walk you through how to actually add this expression to the report. First, select the field in the report to which you want to apply the expression. In this case, select the treatment name textbox from the report in design mode, as shown in Figure 7-4.

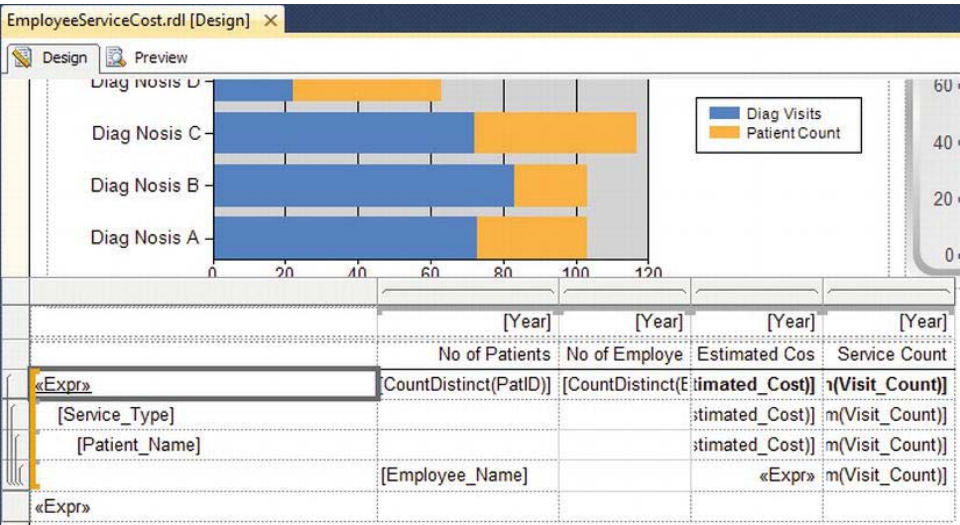


Figure 7-4. Adding the expression to the report

Second, with the textbox selected, go to the Properties window, and select the Color property (see Figure 7-5). If the Properties window is not being shown, you can select View > Properties Window to make it visible from within Visual Studio.

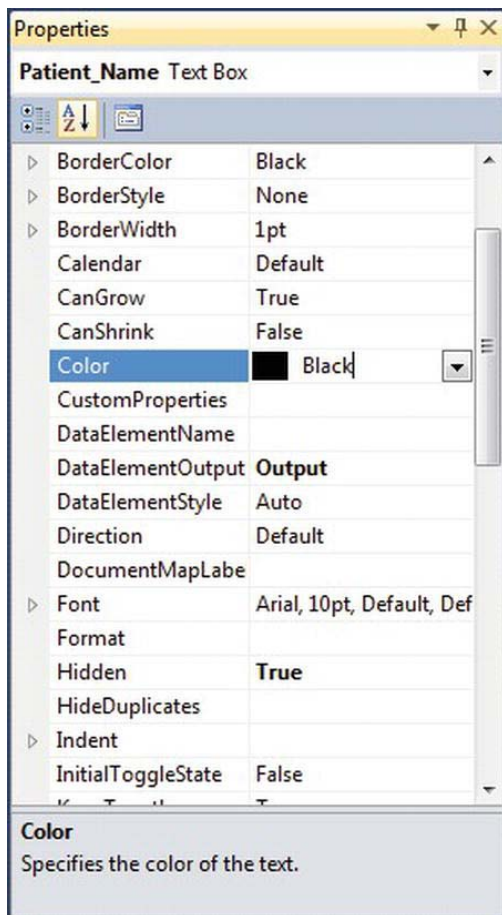


Figure 7-5. Color property in the Properties window

Next, click the down arrow, and from the displayed menu select Expression (see Figure 7-6).

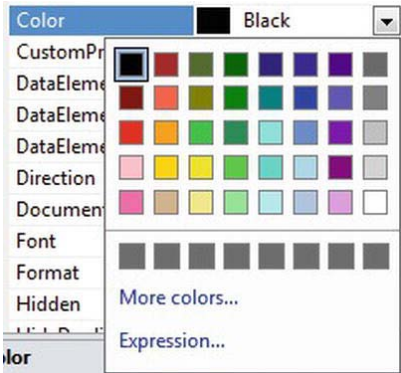


Figure 7-6. Color selection list

Now you will see the Edit Expression dialog box, as shown in Figure 7-7. Enter the expression using the expression code from listing 7-2 here. You can also just type in the expression, or you can use the expression editor to insert the parameters that you need into your expression.

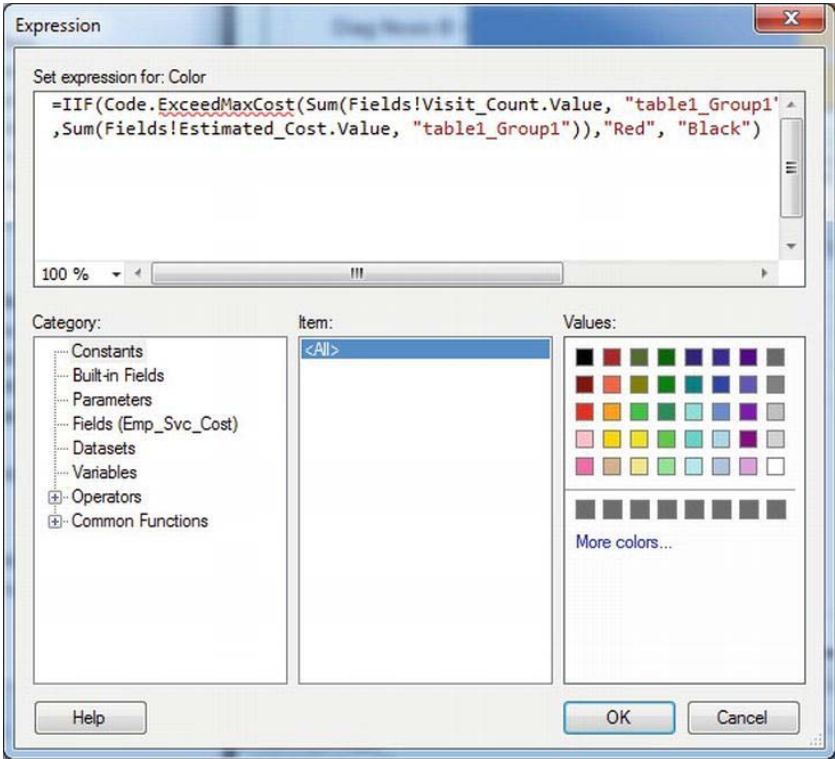


Figure 7-7. Entering an expression in the expression editor

You can now run your report, and the patient name will be displayed in red or black according to the business logic in the Code element of the report.

■ **Note** You will notice that the `ExceedMaxCost` function has a validation error when viewed through the expression editor. This is normal and will not affect your report when building or viewing it.

Now that you have modified the report to use the `ExceedMaxCost` function, you can preview the report to see it in action. To do this, select the Preview tab, and set the `ServiceYear` parameter to have a value of 2010 while leaving all other parameters with their default values. Once you render the report, you should now see one that looks similar to Figure 7-8.

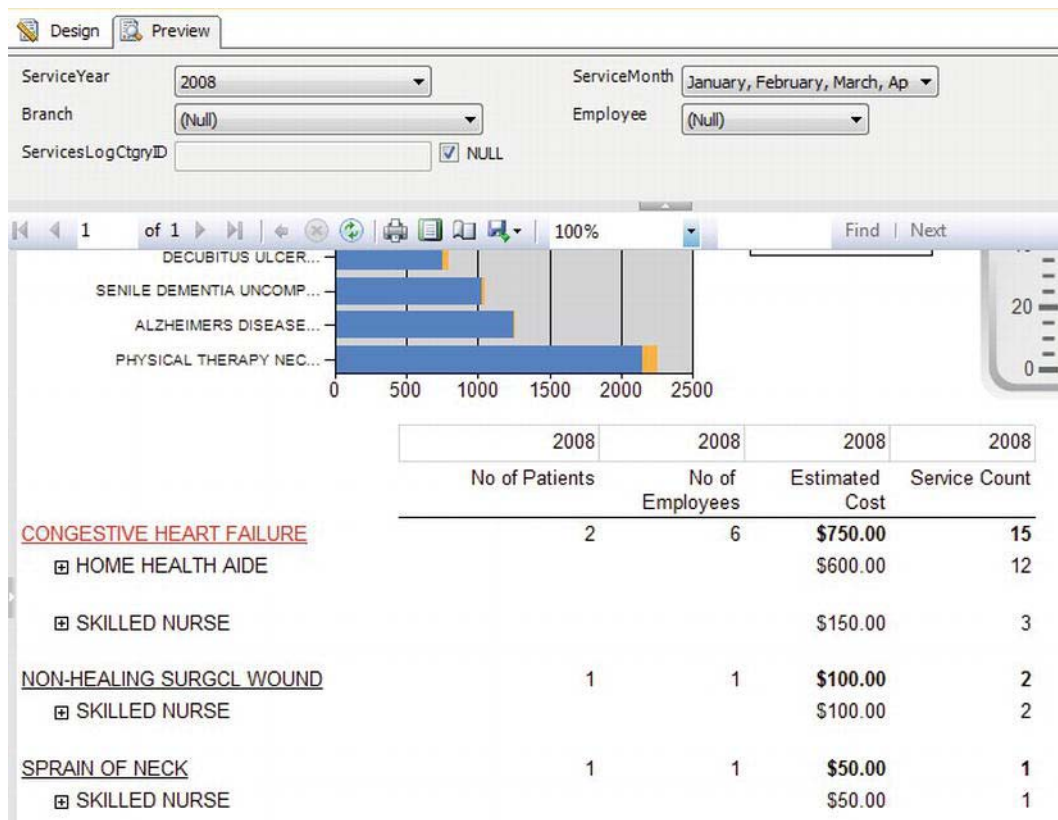


Figure 7-8. Report with the embedded code

Accessing .NET Assemblies from Embedded Code

The Code element of the report was primarily designed for basic use of the .NET framework and VB .NET language syntax. Access to many of the framework namespaces is not included by default in the Code element. Referencing many of the standard .NET assemblies in your embedded custom code requires that you create a reference to each assembly in the report. To do this, go to the References tab of the Report Properties dialog box, click the ellipsis after pressing the Add button, and then select the appropriate .NET assembly you want to reference. Note that, by default, these referenced assemblies will have only Execution permission.

Although it is possible to use other .NET framework assemblies and third-party assemblies directly within the Code element of the report, as just described, it is highly recommended that you consider using a custom assembly instead. One of the primary reasons for this is security. By default, the Code element runs with Execution permission only, which means that it can run but cannot access protected resources. If you need to perform certain protected operations, such as reading data from a file, you'll have to set the security policy for the code group named `Report_Expressions_Default_Permissions` to `FullTrust`. This code group controls permissions for the report expression host assembly, which is an assembly that is created from all the expressions found within a report and is stored as part of the compiled report. To set the security policy, you need to edit the policy configuration files of the report server and the Report Designer. See the “Deploying a Custom Assembly” section later in this chapter for the standard location of these files.

However, making this change to the security policy is not recommended. When you change the permissions for the code that runs in the Code element, you also change the permissions for all reports that run on that report server. By changing permissions to `FullTrust`, you enable all expressions used in reports to make protected system calls. This will essentially give anyone who can upload a report to your report server complete access to your system.

If you need to use features outside the VB .NET language syntax, need additional security permissions, have complicated logic to implement, need to use more of the .NET framework, or want to use the same functionality within multiple reports, then you should move your code into a custom assembly. You can then reference that assembly in your report and use the code through methods and properties of your custom class. Not only does a custom assembly allow you a lot more flexibility in the code itself, it also allows you to control security at a much more granular level. With a custom assembly, you can add a permission set and code group for your custom code to that specific assembly only, without having to modify the permissions for all code that runs in the Code element.

You'll also want to use custom assemblies for another reason. With embedded code, you do not have the benefit of developing the Code section of your report using the full Visual Studio IDE with features such as IntelliSense and debugging at your disposal. Writing code in the Code section of your report is not much different from working in Notepad.

However, you can work around this. If the code you choose to place in the Code element is more than just a few simple lines, it can be easier to create a separate project within your report solution to write and test your code. A quick VB .NET Windows Forms or console project can provide the ideal way to write the code you intend to embed in your report. You get the full features of the IDE, and once you have the methods working the way you want, you can just paste them into the code window of the report. Remember to use a VB .NET project, since the Code element works only with code written in VB .NET.

Using Custom Assemblies with Your Report

Custom assemblies are harder to implement but offer you greater flexibility than embedded code. The process of creating them is a bit more involved because they are not part of the report's RDL and must be created outside the Report Designer. This also makes them more difficult to deploy because, unlike the embedded code, which becomes a part of the report's RDL, the custom assembly is a separate file.

However, your hard work is repaid in many ways.

Code reuse: You can reuse custom code across multiple reports without needing to copy and paste code into each report. This allows you to centralize all the custom logic into a single location, making code maintenance much simpler.

Task separation: Using assemblies allows you to more easily separate the tasks of writing a report from creating the custom code. This is somewhat similar in concept to writing an ASP.NET application using the code-behind feature. This allows ASP.NET developers to separate the page markup, layout, and graphics from the code that will interact with it. If you have several people involved, you can let those who specialize in report writing handle the layout and creation of the report while others who may have more coding skills write the custom code.

Language neutrality: You can use the .NET language of your choice. Choose from C#, VB, J#, or any third-party language that is compatible with the .NET Framework.

Productive development environment: If you use Visual Studio 2010 to develop your custom assemblies, you get the full power of its editing and debugging features.

Security control: You can exercise fine-grained control over what your assembly can do using security policies.

Flexibility: The assemblies that you create are not limited to simple functions in the embedded code section of an RDL. You have the ability to utilize the entire .NET library in your reports.

To use a custom assembly from within your report, you need to create a class library to hold the code, add the methods and properties you want to use from the report to your class, and then compile it into an assembly. To use it from within the Report Designer, you can add a class library project to the solution in Visual Studio, allowing you easy access to both the report and the code you will use in it. Before you run the included examples, make sure to read the `ReadMe.htm` file in the solution's Solution Items folder to see whether any steps are required before running the examples for your particular configuration.

Adding a Class Library Project to Your Reporting Solution

To use a custom assembly with your report, you will first need to write your custom code in the form of a .NET class. You can do this by adding a class library project to your existing solution so that you can work on the report and custom code at the same time.

For this example, you want to display the amount an employee is paid for a visit to a patient. The class will get this information from an XML file that is periodically exported from the human resources (HR) system.

■ **Note** If possible, you would want to get this information directly from the HR system, possibly through a web service. The sample code included with this chapter includes a sample web service and a method to call it.

Using the XML file `EmployeePay.xml` (supplied as part of the code download for this chapter) in this example allows you not only to write a custom assembly, but also to see the steps necessary to access a protected resource such as a local file. To get the information from the XML file and make it available to your report, create a class with a method that takes `EmployeeID` and a date as a parameter and that will read the employee pay per visit rates from the XML file and then return the pay rate. Although we will not cover it step by step in this chapter, the sample code included also contains an example of doing the same thing using a web service. This allows you to simulate being able to interact with the HR system via a web service instead of an exported file.

You can then reference the assembly that you have developed from an expression in the report and use it to calculate the total visiting costs per patient.

To start, select **File > Add > New Project** from the menu. Pick **Visual Basic Projects** or **Visual C# Projects**, depending on your preference. Select **Class Library**, and enter **Employee** for the name of the project. In this example, we will show you how to use a Visual C# class library project, as shown in Figure 7-9.

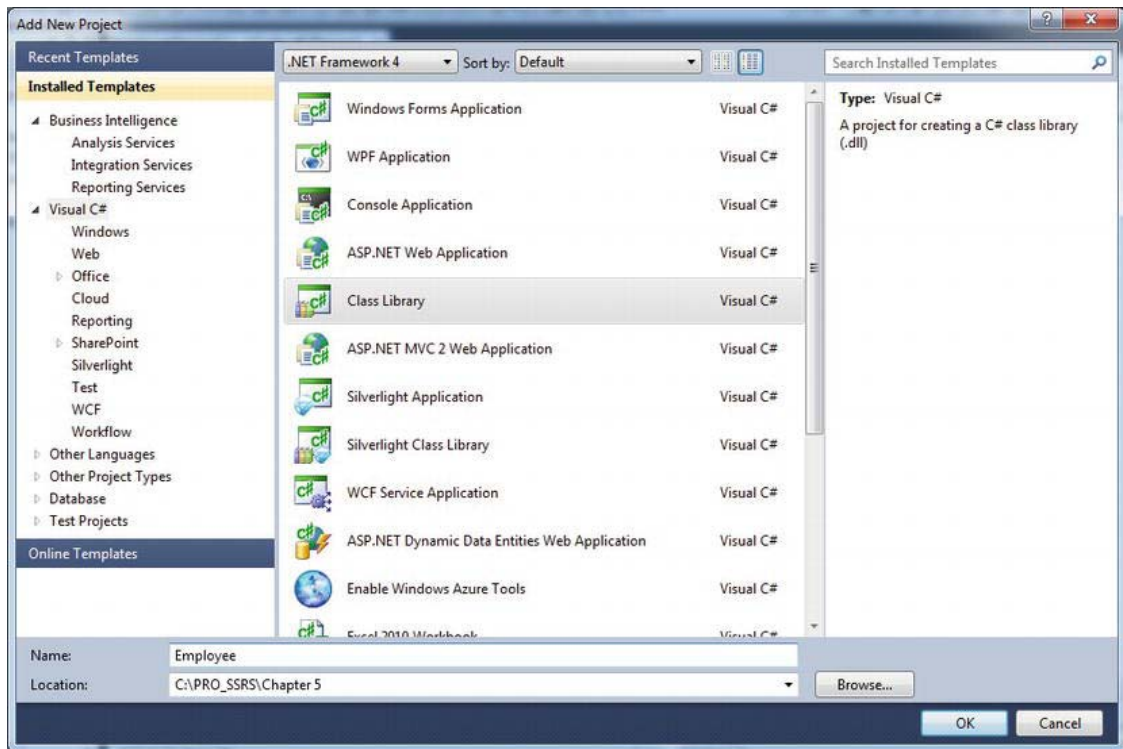


Figure 7-9. Add New Project dialog box

Select the newly created Class1.cs from the solution explorer, and rename it to something a bit more descriptive, such as Employee, because you will use this class to calculate the cost of a visit provided by the employees. Open the Employee.cs file in the Visual Studio 2010 IDE, change the namespace from Employee to Pro_SSRS, and you will see the code editor, as shown in Figure 7-10

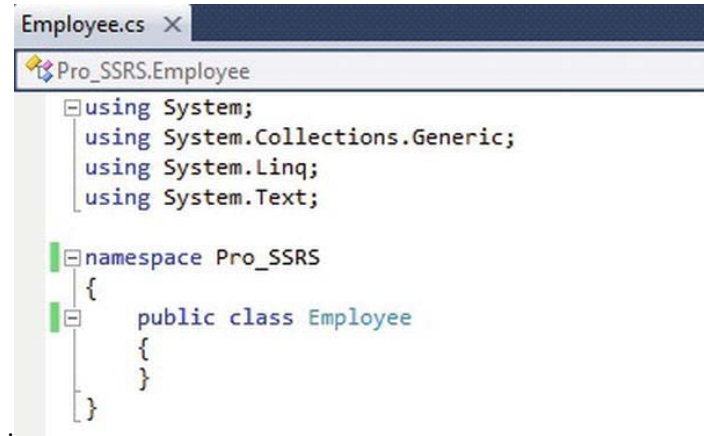


Figure 7-10. The Visual Studio 2008 code editor

For this example, you'll add a few using statements to import types defined in other namespaces. Specifically, you'll add the System.Data and System.Security.Permissions namespaces so you can reference the DataSet and SecurityAction methods without typing in the full namespace in the Employee assembly, as shown in Listing 7-3. You can also drop the reference to System.Linq since we will not be utilizing that in this class library.

Listing 7-3. The Employee Assembly

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Permissions;
using System.Data;

namespace Pro_SSRS {
    public class Employee {
        public Employee() { }

        [PermissionSetAttribute(SecurityAction.Assert, Unrestricted = true)]
        public static decimal CostPerVisitXML(string employeeID, DateTime visitDate)
        {
            DataSet empDS = new DataSet();
            empDS.ReadXmlSchema(@"C:\Temp\EmployeePay.xsd");
            empDS.ReadXml(@"C:\Temp\EmployeePay.xml");
            DataRow[] empRows = empDS.Tables["EmployeePay"].Select("EmployeeID = '" +
employeeID + "'");
            Decimal empAmt;
```

```

        if (empRows.Length > 0)
        {
            empAmt = Convert.ToDecimal(empRows[0]["Amount"]);
            return empAmt;
        }
        else
            return 0;
    }
}

```

Any assemblies used by the custom assembly must be available both on the computer being used to design the report and on the SSRS 2012 server itself. Since you are just using common .NET framework assemblies, this should not be a problem because the .NET Framework is installed on your local computer as well as on the SSRS 2012 server. If you reference other custom or third-party assemblies in your custom assembly, you need to make sure they are available on the SSRS server where you will be running your report.

■ **Note** Because this book's focus is on SSRS and not on writing code, we won't explain the code samples line by line. If you are interested in programming, Apress offers many excellent books for the various programming languages that can help you write custom code for SSRS 2012. Refer to www.apress.com.

To use the Employee assembly in your report, you first need to deploy it to the appropriate location. In the next section, you will learn how to deploy custom assemblies and set up the required permissions. Once you have done that, you will return to the report and use the custom assembly you have created and deployed in the Employee Service Cost report.

■ **Note** Remember that each time you make a change to your custom assembly, you must redeploy the assembly. Also, if you added code that requires additional permissions, you may have to grant them.

Deploying a Custom Assembly

Custom assemblies are more difficult to deploy than code embedded in your report through the Code element. This is because of the following:

- Custom assemblies are not part of the report itself and must be deployed separately.
- Custom assemblies are not deployed to the same folder as the reports.
- The built-in project deployment method in Visual Studio 2010 will not automatically deploy your custom assemblies.

- Custom assemblies are granted only Execution permissions by default. Execution permission allows code to run but not to use protected resources.

To use your custom assemblies with SSRS 2012, you need to take the following steps to place them in a location where SSRS 2012 can find them and to edit the files that control security policy when necessary. The location of the files depends on whether you want to use them in the Report Designer within Visual Studio or on the report server.

1. You need to deploy your custom assemblies to the Report Designer and/or SSRS 2012 applications folder.
- For the Report Designer/Visual Studio, the default is C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies_
- For SSRS 2012, the default is C:\Program Files\Microsoft SQL Server\MSRS11.MSSQLSERVER\Reporting Services\ReportServer\bin_

■ **Note** You need to have the necessary permissions to access these folders. By default, members of the standard Users group won't have the necessary write/modify permissions on these folders. Logged in as a user with the appropriate security permissions, such as the administrator, you can set the permissions on the folders to allow the necessary access to the folders to users logged in under less privileged accounts. Alternately, you could move the files to the appropriate folders when logged in or running as a user who has the necessary permissions.

2. Next, you need to edit the SSRS 2012 security policy configuration files if your custom assembly requires permissions in addition to the Execution permission. (For SSRS 2012, the default location is C:\Program Files\Microsoft SQL Server\MSRS11.MSSQLSERVER\Reporting Services\ReportServer\rssrvpolicy.config.) You should update the designer permissions configuration at this point as well. The default location for this file is C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies\RSPreviewPolicy.config. This config file will use a slightly different assembly path than your SSRS instance, though, since we will be placing the DLL file in the private assemblies folder for Visual Studio 2010.

For example, if you were writing a custom assembly to calculate an employee's cost per visit, you might need to read the pay rates from a file. To retrieve the rate information, you would need to grant additional security permissions to your custom assembly. To give your custom assembly FullTrust permission, you can add the XML text shown in Listing 7-4 to the appropriate CodeGroup section of the rssrvpolicy.config file.

Listing 7-4. *Granting FullTrust Permission to the Custom Assembly for both SSRS and Visual Studio*

```
<CodeGroup class="UnionCodeGroup"
  version="1"
  PermissionSetName="FullTrust"
  Name="EmployeePayCodeGroup"
  Description="Employee Cost Per Visit">
  <IMembershipCondition
    class="UrlMembershipCondition"
    version="1"
    Url="C:\Program Files\Microsoft SQL Server\MSRS11.MSSQLSERVER\Reporting
Services\ReportServer\bin\Employee.dll "/>
</CodeGroup>
```

Or, for Visual Studio:

```
<CodeGroup class="UnionCodeGroup" version="1"
  PermissionSetName="FullTrust"
  Name="EmployeePayCodeGroup"
  Description="Employee Cost Per Visit">
  <IMembershipCondition
    class="UrlMembershipCondition" version="1"
    url="C:\Program Files (x86)\Microsoft Visual Studio
10.0\Common7\IDE\PrivateAssemblies\Employee.dll"/>
</CodeGroup>
```

■ **Note** If you run your report and see #Error text in a textbox instead of the expected result, it is more than likely a permission problem of some kind.

Because it's generally not a good idea to grant your assemblies FullTrust unless absolutely necessary, you can use named permission sets to grant your custom assembly just the permissions it needs rather than FullTrust.

To grant the custom assembly just enough permission to read the data files called C:\Temp\EmployeePay.xml and C:\Temp\EmployeePay.xsd, you first need to add a named permission set in the policy configuration file `rssrvpolicy.config` that grants read permission to the files. This will be placed into the configuration file in the NamedPermissionSets section. You can then apply the specific permission sets to the custom assembly, as shown in Listing 7-5.

Listing 7-5. Named Permission Sets for Reading Files

```

<PermissionSet
  class="NamedPermissionSet"
  version="1"
  Name="EmployeePayFilePermissionSet"
  Description="Permission set that grants read access to my employee cost file.">
  <IPermission
    class="FileIOPermission"
    version="1"
    Read="C:\Temp\EmployeePay.xml" />
  <IPermission
    class="FileIOPermission"
    version="1"
    Read="C:\Temp\EmployeePay.xsd" />
  <IPermission
    class="SecurityPermission"
    version="1"
    Flags="Execution, Assertion" />
</PermissionSet>

```

Next, as shown in Listing 7-6, you add a code group that grants the assembly the additional permissions to the CodeGroup section of the policy configuration file `rssrvpolicy.config`. If you have already added the FullTrust permission to the assembly, you can replace it with this CodeGroup instead, or simply modify it to use the newly created permission set name.

Listing 7-6. Granting File I/O Permission on the Employee Assembly

```

<CodeGroup
  class="UnionCodeGroup"
  version="1"
  PermissionSetName=" EmployeePayFilePermissionSet "
  Name="EmployeePayCodeGroup"
  Description="Employee Cost Per Visit">
  <IMembershipCondition
    class="UrlMembershipCondition"
    version="1"
    Url="C:\Program Files\Microsoft SQL Server\MSRS11.MSSQLSERVER\Reporting
    Services\ReportServer\bin\Employee.dll" />
</CodeGroup>

```

■ **Note** The name of the assembly that you add to the configuration file must match the name that is added to the RDL under the CodeModules element. This is the name you set for the custom assembly under the Report Properties ► References menu, which was introduced in the “Accessing .NET Assemblies from Embedded Code” section; it is discussed in detail in the “Adding an Assembly Reference to a Report” section.

To apply custom permissions, you must also assert the permission within your code. For example, if you want to add read-only access to the XML files `C:\Temp\EmployeePay.xsd` and `C:\Temp\EmployeePay.xml`, you must add code similar to that shown in Listing 7-7 to your method.

Listing 7-7. Asserting Permission with Code

```
// C#
FileIOPermission permissionXSD = new
    FileIOPermission(FileIOPermissionAccess.Read,
        @" C:\Temp\EmployeePay.xml");
permissionXSD.Assert();
// Load the schema file
empDS.ReadXmlSchema(@"C:\Temp\EmployeePay.xsd");

FileIOPermission permissionXML = new
    FileIOPermission(FileIOPermissionAccess.Read, @"
C:\Temp\EmployeePay.xml");
permissionXML.Assert();
empDS.ReadXml(@"C:\Temp\EmployeePay.xml");
```

You can also add the assertion as a method attribute, as shown in Listing 7-8. This is the method shown in this chapter's examples.

Listing 7-8. Asserting Permission with a Method Attribute

```
[FileIOPermissionAttribute(SecurityAction.Assert,
    Read=@" C:\Temp\EmployeePay.xsd")]
[FileIOPermissionAttribute(SecurityAction.Assert,
    Read=@" C:\Temp\EmployeePay.xml")]
```

■ **Tip** For more information about code access security and reporting services, see “Understanding Code Access Security in Reporting Services” in the SSRS 2012 Books Online (BOL). For more information about security, see “.NET Framework Security” in the .NET Framework Developer’s Guide, available on the Microsoft Developer Network (MSDN) web site at <http://msdn.microsoft.com>. You will also want to read about using the Global Assembly Cache (GAC) to store your custom assembly.

Adding an Assembly Reference to a Report

With the `EmployeeServiceCost-NoCode` report selected and on the Design tab, select **Report ► Report Properties** from the top menu items; alternatively, right-click within the report design area, and select **Properties**. Then do the following:

1. With the References section active, click the Add button in the assemblies section.
2. Click the ellipsis button, and then click the Browse tab. Browse to the location of the Employee.dll file and select it. If you have already placed the built DLL in the private assemblies folder for Visual Studio, you should do this now so you can reference that location. When you are done, the Report Properties dialog box should look like Figure 7-11.

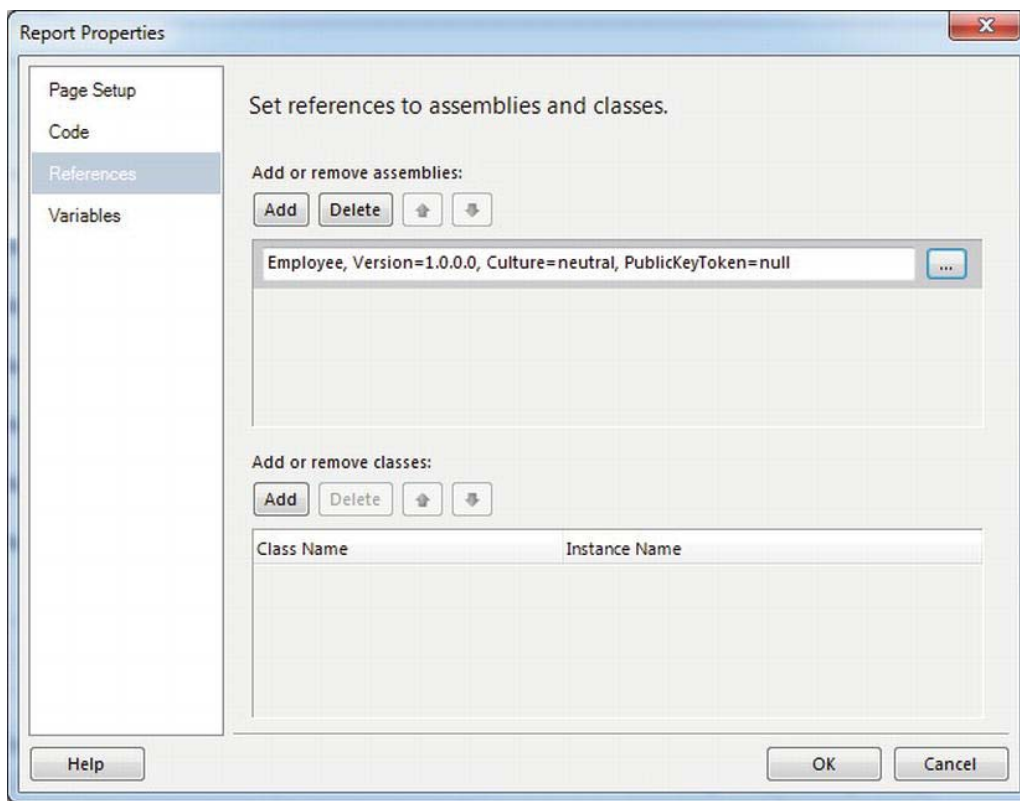


Figure 7-11. References tab

■ **Note** The class list on the References tab of the Report Properties dialog box is used only by instance-based members, not static members.

To use the custom code in your assembly in a report expression, you must call a member of a class within the assembly. You can do this in different ways depending on how you declared the method.

If the method is defined as static, it is available globally within the report. You access it in an expression by the namespace, class, and method name. The following example calls the static `CostPerVisit` method in the `Employee` class, which is in the `Pro_SSRS` namespace, passing in an `EmployeeID` value and the visit date. The method will return the cost per visit for the specified employee.

```
=Pro_SSRS.Employee.CostPerVisitXML(empID, visitDate)
```

If the custom assembly contains instance methods, you must add the class and instance name information to the report references. You do not need to add this information for static methods.

Instance vs. Static Methods

Static methods are methods that can be called from a class that do not act on a specific instance of the class. This means that a new object does not need to be instantiated for the method to be called and used. Instance methods do require an object to be created so they can be used and will perform any work they do in that specific object of that type of class, or instance, of the class. Static methods are preferable here since we don't need to do any additional work in order to create a new object of that class type to begin using the methods.

Instance-based methods are available through the globally defined `Code` member. You access these methods by referring to the `Code` member and then the instance and method name. The following shows how you would call the `CostPerVisitXML` method if it had been declared as an instance method instead of a static method:

```
=Code.Employee.CostPerVisitXML(empID, visitDate)
```

■ **Tip** Use static methods whenever possible because they offer higher performance than instance methods. However, be careful if you use static fields and properties, because they expose their data to all instances of the same report, making it possible that the data used by one user running a report is exposed to another user running the same report.

After adding the reference to the `Employee` custom assembly, you will use it by calling the `CostPerVisitXML` method as a part of an expression in the report. Highlight the `Employee_Cost` textbox in the report, as shown in Figure 7-12.

		[Year]	[Year]	[Year]	[Year]
		No of Patients	No of Employee	Estimated Cos	Service Count
«Expr»		CountDistinct(PatID)	CountDistinct(E	Estimated_Cost)	m(Visit_Count)
[Service_Type]				Estimated_Cost)	m(Visit_Count)
[Patient_Name]				Estimated_Cost)	m(Visit_Count)
		[Employee_Name]		Estimated_Cost)	m(Visit_Count)
«Expr»					

Figure 7-12. Employee_Cost textbox

Right-click, select Expression, and in the Edit Expression dialog box enter the code shown in Listing 7-9.

Listing 7-9. Using the CostPerVisitXML Method in an Expression

```
=Pro SSRS.Employee.CostPerVisitXML(Fields!EmployeeID.Value,
Fields!ChargeServiceStartDate.Value)* sum(Fields!Visit_Count.Value)
```

Now if you preview the report or build and deploy it, you should see a report similar to Figure 7-13. You can use any parameters you want to generate an actual report. Once you do, expand both the service type and patient name tree menu items to see the newly calculated cost estimate values.

ServiceYear 2008 ServiceMonth January, February, March, Ap
Branch (Null) Employee (Null)
ServicesLogCtgrID NULL

1 of 1 0 500 1000 1500 2000 2500 100% Find | Next

	2008	2008	2008	2008
	No of Patients	No of Employees	Estimated Cost	Service Count
<u>CONGESTIVE HEART FAILURE</u>	2	6	\$750.00	15
HOME HEALTH AIDE			\$600.00	12
PoundEzra			\$500.00	10
CrachetNurse			\$30.00	1
XmzjcoUdv			\$270.00	9
VgckjlzZfls			\$100.00	2
SKILLED NURSE			\$150.00	3

Figure 7-13. Final report

In the sample code for Chapter 7, we have also included a web service that can be called from the custom code to access the employee pay information. This simulates accessing information from another system via a web service and is designed to allow you to replace the exported XML file with a call to a web service. The Employee class included in the sample code already contains a method called CostPerVisitWS that uses the web service rather than the XML file as its source of data. You can make the report use the web service instead of the XML file by simply changing the expression from this:

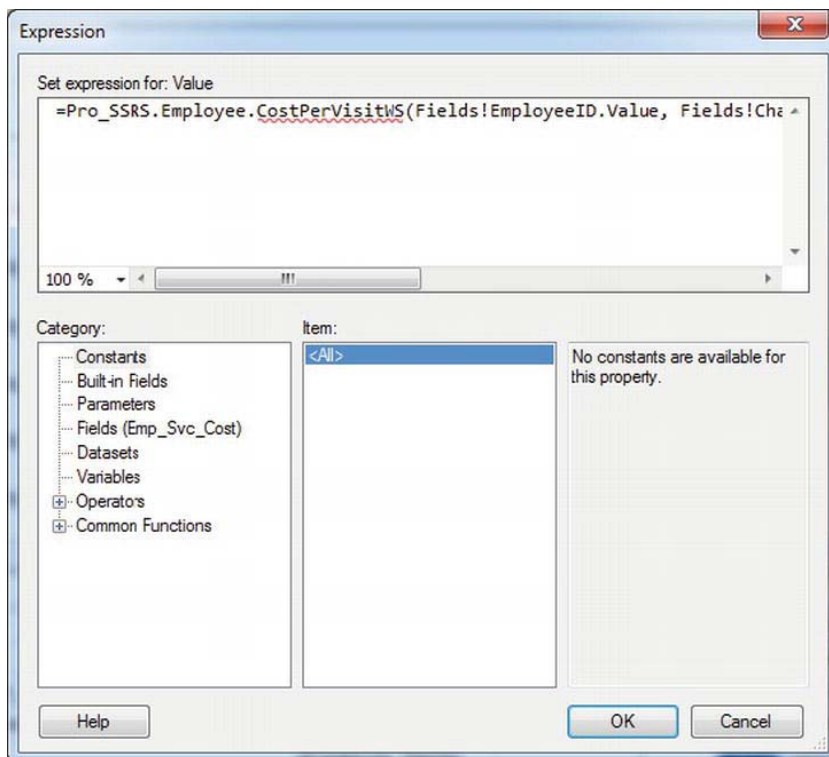
```
=Pro_SSRS.Employee.CostPerVisitXML(Fields!EmployeeID.Value,  
Fields!ChargeServiceStartDate.Value)* sum(Fields!Visit_Count.Value)
```

to the following:

```
=Pro_SSRS.Employee.CostPerVisitWS(Fields!EmployeeID.Value,  
Fields!ChargeServiceStartDate.Value)* sum(Fields!Visit_Count.Value)
```

To try this, just open the sample solution for this chapter, and edit the expression in the Employee_Cost field in the EmployeeServiceCost.rdl file. To try the Employee Web Service, you also must make sure the Web service is running when you debug the report in Visual Studio or that it is published and running on your test machine if you have the report deployed to the server. Let's go through the code in our Employee class that calls the web service and show how the process works in our report.

Open the EmployeeServiceCost report in design mode and go down to the employee estimated cost column in your table. Let's edit the expression using the previous example we provided for use with the web service. Once you edit the expression to use the web service call, it should look like Figure 7-14.



We are not using a separate method that is contained in our Employee class, which would use a web service to pull the data we want rather than use an XML file. Let's take a look at this method from Employee.cs in our Employee class project. You will see in Listing 7-10 that it is just a simple call to the web service that we have built and is included in our project. We provide the web service the employeeID and visit date, just like we did when we pulled the information from XML, but this time the data is provided directly from the web service, which we will look at in just a bit.

Listing 7-10: Getting Cost Values Through a Web Service

```
public static decimal CostPerVisitWS(string employeeID, DateTime visitDate)
{
    EmployeeWS.Employee empWS = new EmployeeWS.Employee();
    Decimal    empAmt;
    empAmt = empWS.GetCost(employeeID, visitDate);
    return    empAmt;
}
```

We will do much more with web services in Chapters 8 and 9 if you are not yet familiar with them. It looks to be a much easier process than when we parsed through the XML file, but the bulk of the work is actually being done on the web service that has been provided in the solution.

Since the web service, in the solution provided, is already set up to have a dependency on the web service we don't need to bother with those to test our new data call. Go ahead and start the debug process, making sure that the reports project is still set up as the startup project in the solution. You can set the parameter of ServiceYear to be 2010 and view the report. Once you drill down into a patient, you will notice that the report processes and refreshes the data. This time the data is being pulled through the web service, which gets the data from the database instead of from an XML file. We have also included a sample test application that allows you to call the CostPerVisitXML and CostPerVisitWS methods of the Employee class using a Windows Forms application. This allows you to exercise the class and step through the code in the Windows Forms environment, which is easier to test and debug.

■ **Tip** Writing a test application is a great way to make sure your custom code is properly performing the expected functions prior to using the code within your SSRS 2012 report. Not only can you create a custom Windows Forms application as we did, but with the proper version of Visual Studio 2010, you can create specialized test code and automated test routines.

Debugging Custom Assemblies

For ease of debugging, the recommended way to design, develop, and test custom assemblies is to create a solution that contains both your test reports and your custom assembly. This will allow you easy access to both the report and the code you will use in it at the same time from within Visual Studio.

We'll now show you how to set up Visual Studio to allow you to debug the Employee assembly you have just written:

1. In the Solution Explorer, right-click the solution, and select Configuration Manager. This will allow you to set the build and deploy options for debugging.

2. Select Debug as the Active Solution Configuration option. Click Close to finalize this option choice.
3. Right-click the project containing your reports from the solution manager and select Set as StartUp Project to make sure this project runs first in your solution when you start to debug.
4. Right-click the project containing your reports from the solution explorer again, and select Project Dependencies.
5. In the Project Dependencies dialog box, select the Employee project as the dependent project, as shown in Figure 7-14. This will tell Visual Studio that your report depends on the custom assembly you have written.

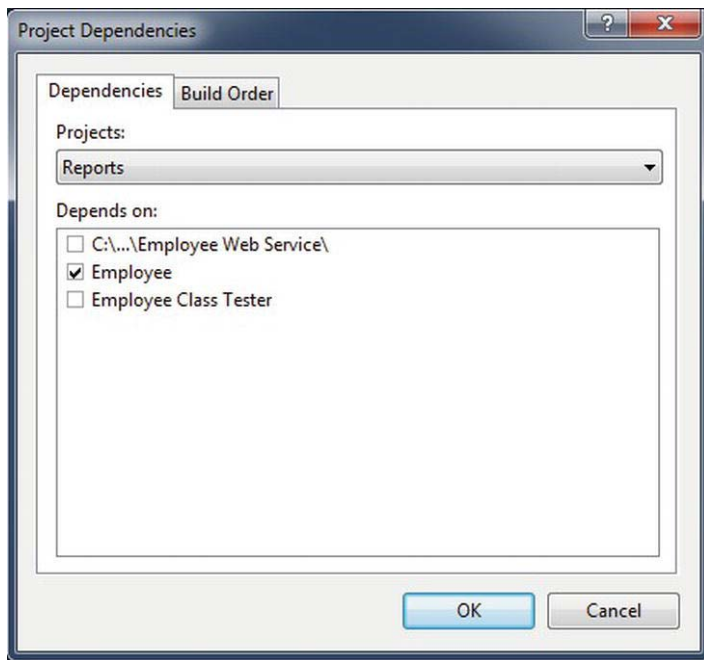


Figure 7-14. Project Dependencies dialog box

6. Click OK to save the changes, and close the Project Dependencies dialog box.
7. Right-click the Reports project again, and select Properties.
8. Select StartItem, and set it to the report you want to debug. In this case, select the report we have set up to use the custom assembly. The StartItem option tells Visual Studio specifically which report to run when you run with debugging. Click OK to finalize this choice.
9. In the Solution Explorer, select the Employee custom assembly project.
10. Right-click this project and select Properties.

11. Select the Build section from the left side menu.
12. On the Build page, enter the path to the Report Designer folder in the Output Path textbox. (By default, this path is C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\PrivateAssemblies.)

■ **Note** You will need to have the necessary permissions in order to use this folder for the output path. By default, members of the standard Users group won't have the necessary write/modify permissions on this folder. When you're logged in as a user with appropriate security permissions, such as the administrator, make sure to set the permissions on the PrivateAssemblies folder to allow the necessary access when you are logged in under a less privileged account. You could leave the default output path, but changing it saves you some work. With the default path, you'd have to build and then manually copy your custom assembly in order for the Report Designer running within Visual Studio to find it and run it.

13. Now set break points in your custom assembly code. If you are not familiar with setting break points, simply open your `employee.cs` file and click in the grey side panel to the left of the code. Set a break point at the point where the method reads the XML data file as seen in Figure 7-15.
14. Make sure to set Report as the startup project, and then press F5 to start the solution in debug mode. When the report uses the custom code in your expression, the debugger will stop at any break points you have set when they are executed. Now you can use all the powerful debugging features of Visual Studio to debug your code.

■ **Note** It is also possible to use multiple copies of Visual Studio to debug your custom assembly. See the SSRS 2012 BOL for details.



Figure 7-15. Setting a break point in your custom assembly

Troubleshooting Your Project

If you modify a custom assembly and rebuild it, you must redeploy it, because the Report Designer looks for it only in the Report Designer application folder. If you followed our suggestion in the “Debugging Custom Assemblies” section to change the output path, it should be in the correct location each time you rebuild it while debugging. If not, you will need to follow the instructions in the “Deploying a Custom Assembly” section to move it to the Report Designer application folder. Remember, Visual Studio will not deploy your custom assembly to your SSRS 2012 server machine; you must copy it manually. Finally, you may want to keep the version of any custom assembly the same value, at least while you are developing it. Every time you change the version of a custom assembly, the reference to it must change on the References tab of the Report Properties dialog box, as discussed earlier in this chapter. Once your reports are in production where you want to keep track of version information, you can use the GAC, which can hold multiple versions; this means you have to redeploy only those reports that use the new features of the new version. If you want all the reports to use the new version, you can set the binding redirect so that all requests for the old assembly are sent to the new assembly. You would need to modify the report server’s `Web.config` file and `Rsreportserver.config` file.

If you are using a custom assembly and the output on your report shows `#Error`, you likely have encountered a permissions issue. See the “Deploying a Custom Assembly” section in this chapter for information on how to properly set up the permissions.

Summary

In this chapter, you learned how to use custom code with your reports, and we discussed some of the other programmatic aspects of dealing with SSRS 2012. Chapters 8, 9, and 10 will build on this as we show you how to write custom applications to render reports, deploy them to the report server, and schedule them to run using subscriptions.

Deploying Reports

Throughout the lifecycle of a report—from creation to maintenance—administrators, developers, and now even end users using Report Builder need to continually deploy reports to the SSRS 2012 server. Deploying a report simply means uploading the RDL file onto the SSRS 2012 server so that your users can use it (for more information on the specifics of the RDL format of these reports, see Chapter 3).

Fortunately, SSRS 2012 provides several means for deploying reports:

Using the Report Manager interface through your Web browser: This simple method allows anyone with an RDL file and the proper SSRS permissions to upload it to the SSRS 2012 server. This can be especially useful if you're developing your report's RDL files in an application that doesn't provide you with a method to upload them to the server. It's also useful if you want to make a quick edit of the RDL file—say to change a misspelled word—using an application such as Notepad, which doesn't offer a built-in way to upload the report. We cover this scenario in Chapter 8, where Notepad is used to modify a report.

Using Report Builder 3.0: The Report Builder 3.0 utility was a new feature to SSRS 2008 R2 and it is still used in SSRS 2012. It provides the user who is unfamiliar with the Visual Studio IDE with a simple interface by which to create and edit data sources and reports.

Using the Deploy option in BIDS/Visual Studio: This method allows you to deploy your reports to the SSRS 2012 server from directly within your development environment. If you're using BIDS/Visual Studio and have direct access to the report server to which you want to deploy your reports, this is one of the easiest options.

Using the rs.exe command-line utility: The rs.exe command-line utility is a runtime environment that is used to execute VB .NET code in the form of specially formatted script files. You deploy the report in the same way as the method in the next item of this list. Later in the section "Using the rs.exe Utility," we will cover how to deploy reports and create data sources programmatically using rs.exe.

Programmatically, using the SSRS Webservice: This method gives you complete control over the deployment process with the added advantage of creating any type of UI you want. Unlike the rs command-line utility, you have your choice of languages and the full power of Visual Studio 2012 to help you develop your custom interface. In Chapter 6, we used the SOAP API, otherwise known as the Report Server Web service, to retrieve report parameter information about

reports from the SSRS 2012 server, and then used that information to generate a Windows Forms UI for parameter selection. In this chapter, you'll use the Report Server Web service to publish reports to the SSRS 2012 server. This type of interface is useful when you need to integrate the deployment of reports into your setup, installation, or runtime environment of your custom application.

Using Report Manager

To deploy reports using the Report Manager interface, simply open your browser and navigate to your SSRS 2012 server using an address such as `http://localhost/reports/`. You'll see a screen similar to the one shown in Figure 8-1.

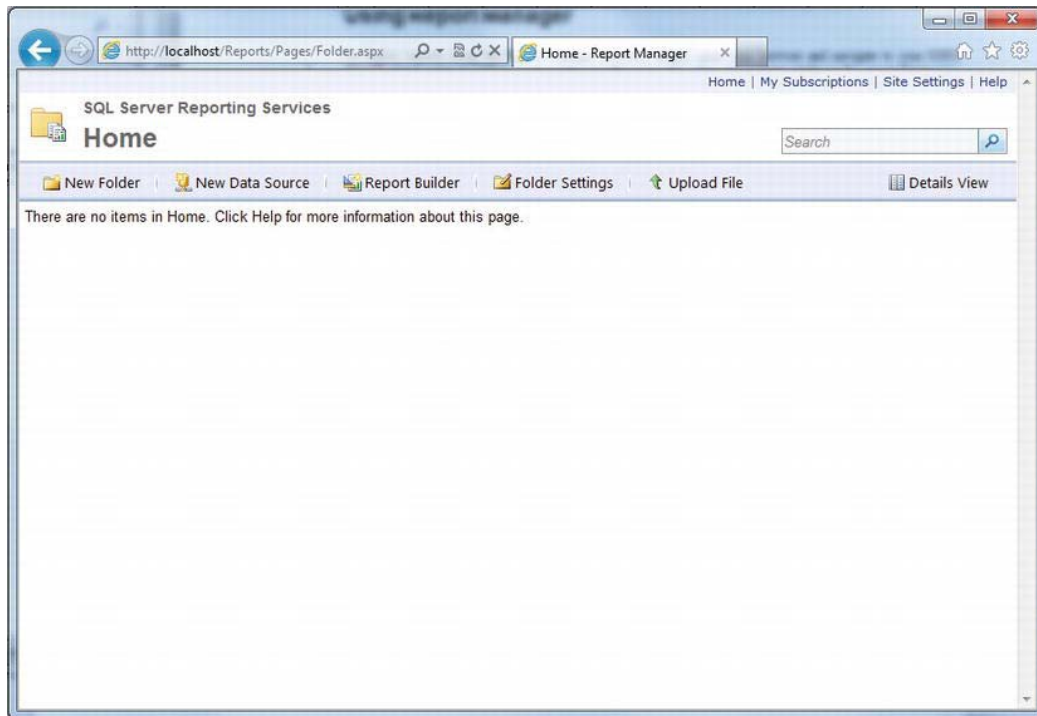


Figure 8-1. Report Manager

As you can see, an Upload File option is available on the Report Manager toolbar. Selecting Upload File opens a standard browser-based upload-style dialog box, such as the one shown in Figure 8-2.

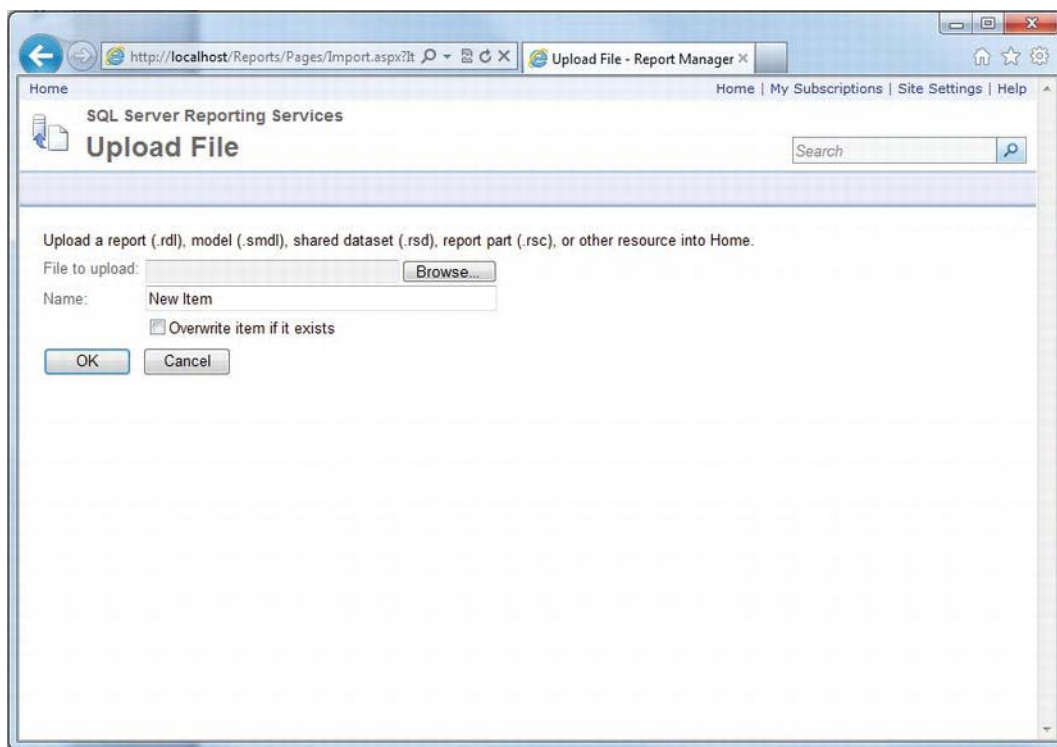


Figure 8-2. Report Manager Upload File dialog box

Using this dialog box, you can browse to the RDL file that you want to upload and then upload it. You can also upload data source files here too, as well as RDS files, but they won't be automatically seen as data sources. We will show you later in the chapter how to create datasources correctly.

Browse to sample files for Chapter 6 and select the `EmployeeServiceCost.rpt` file in the reports folder to upload to your report server. The Report Manager places the file into the current folder (the one from which you initiated the upload process).

Note that newly uploaded reports are no longer tagged with a marker showing that they are new, which was done in some older versions of SSRS. Each report that you upload will have a dropdown menu from within the report manager interface. This menu allows you to perform some management functions and open the properties editing page. This "manage" option can be seen below in Figure 8-3.

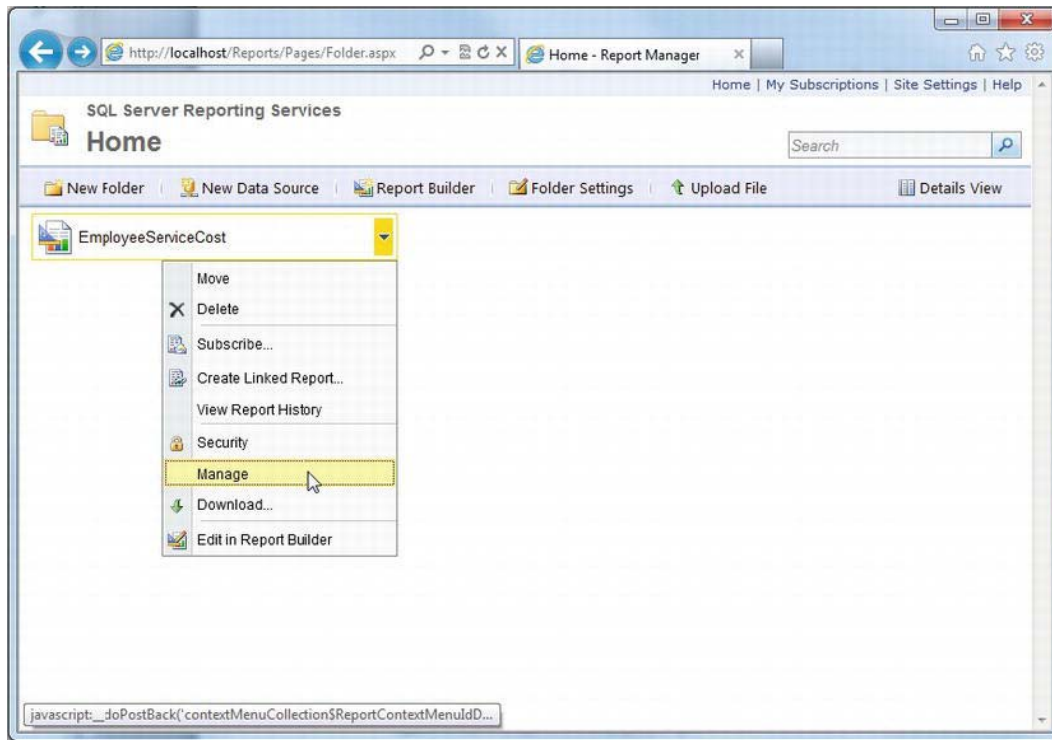


Figure 8-3. Report Manager with a recently uploaded report

This manage option allows you to modify the properties of the report, such as the name and description, as shown in Figure 8-4.

There is a checkbox in the bottom-center of Figure 8-4, just above the Apply button, that allows you to do the following:

Hide the report in the tile view: This can be useful if you don't want users with access to the SSRS 2012 server to know that certain reports exist or if you don't want them to see the detailed information about the report. Remember that this only hides the report in the tile view and not the detail view. You should use SSRS security properties when you need to prevent a user from seeing the contents of folders or running reports they shouldn't have access to.

Then, the menu going across the top of the screen allows you to:

Delete the report: This gives you the ability to remove reports that are no longer needed.

Move the report to another location on your report server: This allows you to organize the reports into folders for organizational and security purposes.

Create a linked report: This option allows you to create a linked report that is based on the original report. This allows you to keep one copy of the report for

deployment somewhere on the server, but have multiple copies available in other locations on the server without the need to redeploy them each time the report is changed.

Download the RDL: You can download the report RDL to a file that you can edit. This can be useful when you want to make a minor modification to a report, such as changing the spelling of a word or modifying an expression. Keep in mind that this method only provides you access to the RDL file; you still have to use another program to edit the file and then you have to upload the modified file.

Replace the RDL by uploading a new copy of the RDL file: If you've downloaded and edited the RDL file, then this option allows you to upload the edited copy.

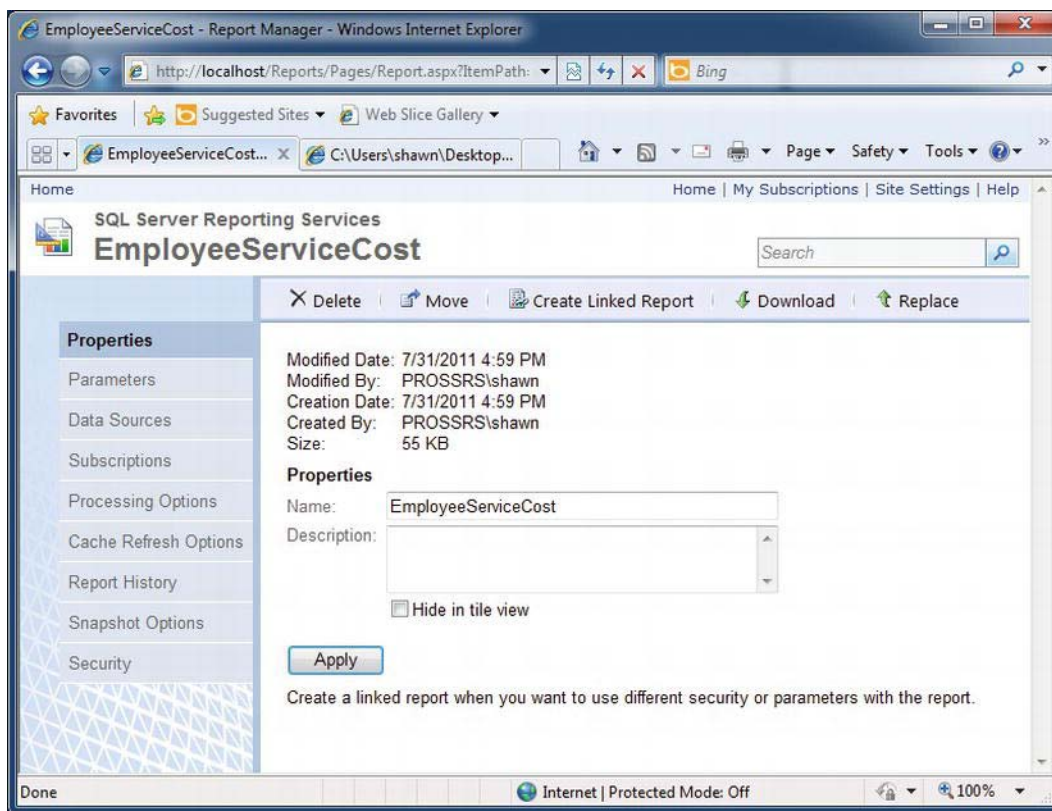


Figure 8-4. Report Properties page

Using the menu on the left side of the page in Figure 8-4, you can do the following:

Make report parameter modifications: From the Parameters section, you can change some options on the report parameters, such as default value, visibility, user prompting, and parameter display text. This ability to change parameters

is convenient, since you do not have to download a report to make simple parameter changes.

Make changes to the data source: The Data Sources section allows you to change the data source used by the report. You can change to another data source located on the same server, or you can create a new custom data source from scratch.

Manage Subscriptions: The subscription sections allow you to manage subscriptions and data driven subscriptions of the report. You can create, delete, and modify your report subscriptions from this section.

Processing Options: This section allows you to configure data caching options for use when rendering reports. It also allows you to set up rendering of the report using a report snapshot and configure the report timeout properties.

Cache Refresh Options: You can set up a new cache refresh plan or make modifications to current plans from this section. These plans will create refresh jobs that update cache stores used for reports or shared data sources. This data will be used if your report is set up to render from cached data and not from live data.

Report History: This section allows you to take snapshots of the report or to view historical snapshots of rendered reports. These can be created manually or by a snapshot schedule that is configurable in the next section.

Snapshot Options: From this section, you can manage snapshots and snapshot schedules. You can create a new snapshot schedule or set up a new snapshot on a shared schedule. You also have the ability to manage options for snapshot security and snapshot storage limits.

Manage security settings: You can manage individual security settings for a report from the Security section of the page. By default, a report inherits the parent folder's security settings, but you can break that inheritance (or restore it) from this menu.

■ **Note** You can access the management section from the detail view as well.

Using Report Builder 3.0

Another method that is used to deploy reports in Reporting Services 2010 is the new Report Builder 3.0 application. Report Builder 3.0 is a more user friendly way to deploy reports that moves away from using BIDS/Visual studio. We describe using the new BIDS/Visual Studio integrated tools in the next section. You can launch the Report Builder tool from the report manager web interface. You can also install Report Builder 3.0 as a separate utility by downloading the full installer from the Microsoft website. Figure 8-5 will give you an idea of what to expect visually from this new utility.

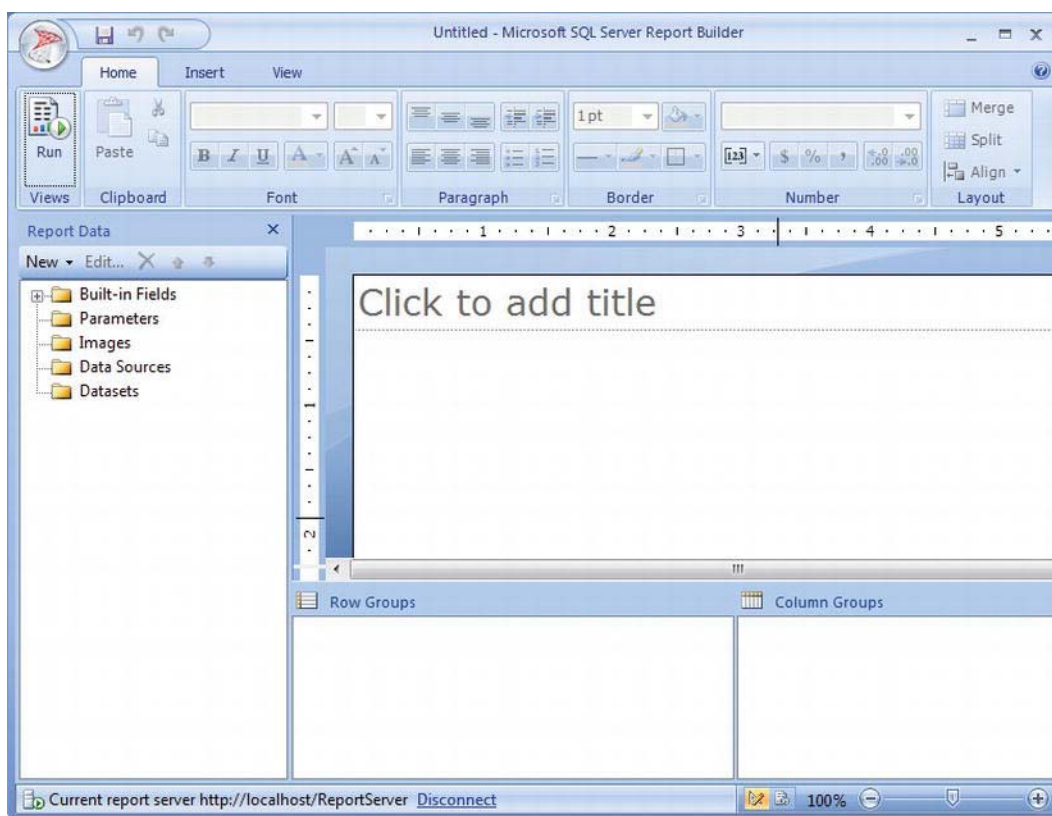


Figure 8-5. The new Report Builder 3.0 interface

Report Builder 3.0 will feel more familiar to users of Windows 7 or the newer Office suites. The main menu for this program is located under the Pearl in the top-left corner of the screen that looks like the SQL Server symbol. Click the Pearl once to expand the menu and select Open. This will allow you to browse to the report that you wish to open for editing or deploying. This report can be loaded from a file on a local/remote disk or directly from the report server that you are connected to. Once the file has been loaded, the report will appear in design mode in the center window, and the data will populate the window to the left. From here, you can make any changes needed before deploying to the server. Since you opened the report builder from the web interface, go ahead and load the report from the server and your Report Builder should resemble Figure 8-6.

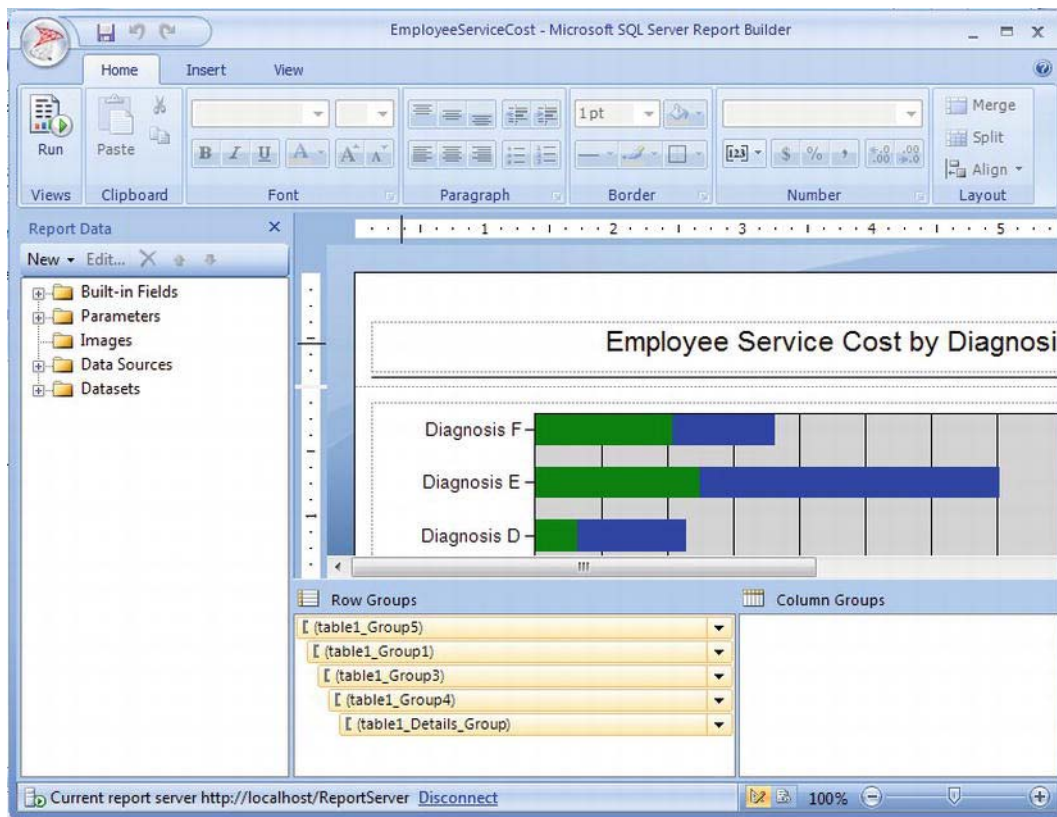


Figure 8-6. Loaded report in Report Builder 3.0

You have the ability to perform most of the functionality that is included with BIDS/Visual Studio 2010 in Report Builder 3.0. From this program, you can add data sources, make report design changes, manage parameters, and deploy reports. The only drawback is that this application only allows for one report to be modified at a time. To perform mass deployments or manage multiple reports for a project, you would still need to use Visual Studio 2010/BIDS or the RS utility, which we will cover later in this chapter. To deploy the report you have just opened, and assuming that you have opened the report from the report server directly, return to the Pearl menu and choose Save. If you have opened the RDL file from your hard drive, then use the Save As option and make sure that you are connected to a report server first. To connect to a report server, simply click the connect link in the bottom status bar of the Report Builder 3.0 interface. Once you are connected to your report server, you can choose that server as the destination for your report to be saved. See Figure 8-7 to see the Pearl menu options that you will be using. Since we launched Report Builder from the web interface directly, we are already connected to the report server and will not be required to specify a server to publish to. Go ahead and make a small change to your report and deploy it to see the changes on the report manager web interface.

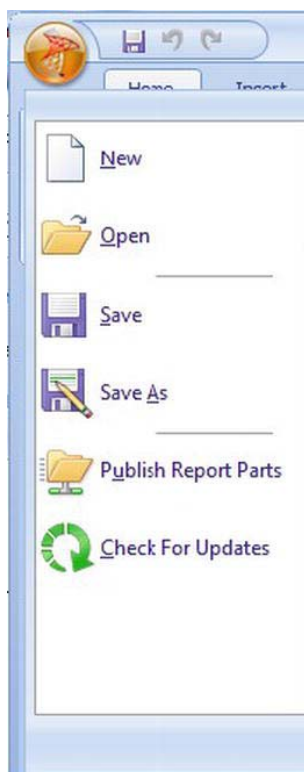


Figure 8-7. Report Builder 3.0 save options

Using BIDS and Visual Studio 2012

You also can deploy reports using the Deploy option in BIDS (Business Intelligence Development Studio), which is integrated directly into Visual Studio 2010. This is convenient, because it is the most robust environment and you will be developing your reports with this tool in most cases.

Configuring Report Deployment Options

BIDS/Visual Studio 2010 allows you to configure a different set of properties for each project in your solution. You also can set these properties for each configuration available for your project, such as Debug, DebugLocal, and Release. For each configuration of a report project, you can define values uniquely for the following properties:

- **Start Item:** The name of the report to be displayed in the preview window or in a browser window when the report project is run.
- **OverwriteDataSets:** A Boolean value that configures whether to overwrite existing data set definitions on the report server. Setting this value to true will overwrite any data set definitions when deploying.

- **OverwriteDataSources:** A Boolean value indicating whether to overwrite an existing data source on the server. Set it to true to overwrite, which redeploys any data sources you have defined in your project each time you select Deploy. Set it to false if you don't want existing data sources to be overwritten.
- **TargetDataSetFolder:** The name of the folder where your shared datasets, whether a report server path or a SharePoint library, are placed.
- **TargetDataSourceFolder:** The name of the folder in which to place your shared data sources.
- **TargetReportFolder:** The name of the folder in which to place your reports. By default, this is the name of the report project.
- **TargetReportPartFolder:** The name of the folder in which to place your report parts that are shared with other reports.
- **TargetServerURL:** The URL of the target report server, such as `http://localhost/reportserver`.
- **TargetServerVersion:** This option allows you to specify what version of SSRS the deployment server is running. You can also have this option auto-detect the version if you have already configured the target server URL.

The different configurations are convenient for the report developer because you can set up different servers and/or folders for testing and deployment in the same project. By default, when you create a report project, BIDS/Visual Studio 2010 creates three different configurations for you: Debug, DebugLocal, and Production. You can access the properties of these configurations through the project's Property Pages. To get to the configuration Property Pages, in the Solution Explorer, right-click the project containing the reports and then click Properties. To see the property settings for each configuration option, select from the Configuration drop-down list at the top of the properties dialog box. Figure 8-8 shows the Property Pages for the Reports project in the Chapter 6 solution that is part of the sample code provided for this chapter in the Source Code/Download area of the Apress Web site (<http://www.apress.com>).

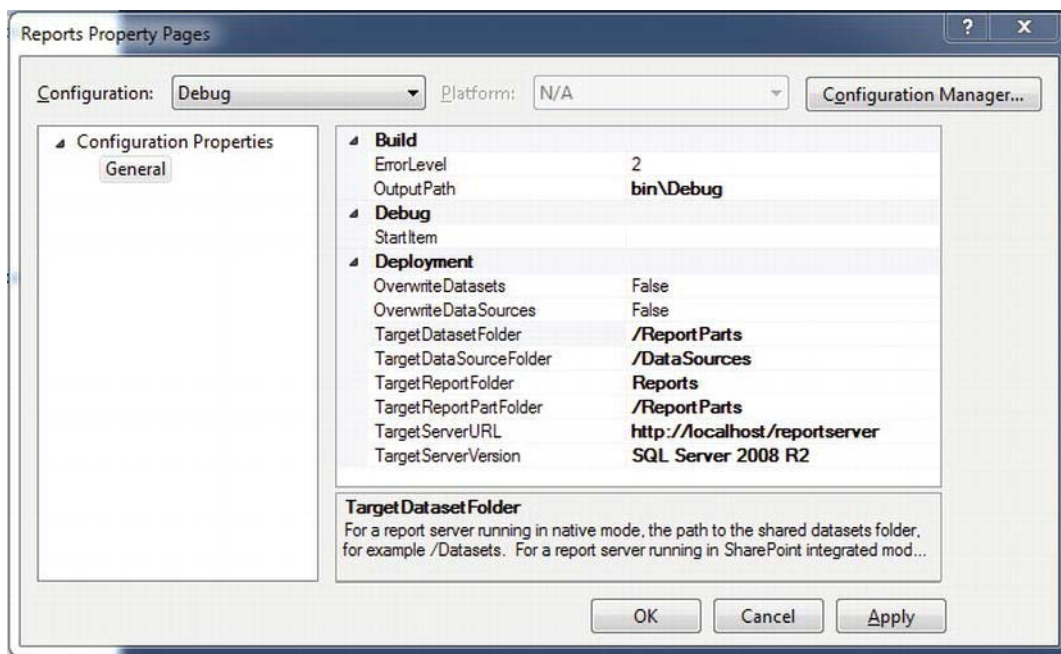


Figure 8-8. Project Property Pages

After the configuration information is set up correctly, you can deploy reports to your server by using the Build and Deploy options in the Configuration Manager or by using the Solution Explorer.

Setting Up Deployments Using the Configuration Manager

The Build and Deploy options in the Configuration Manager determine whether reports are built, deployed, or both when you start the project in Visual Studio 2010.

To open the Configuration Manager, in the Solution Explorer, right-click the project containing the reports and then click Properties. From there, click Configuration Manager to open the dialog box shown in Figure 8-9.

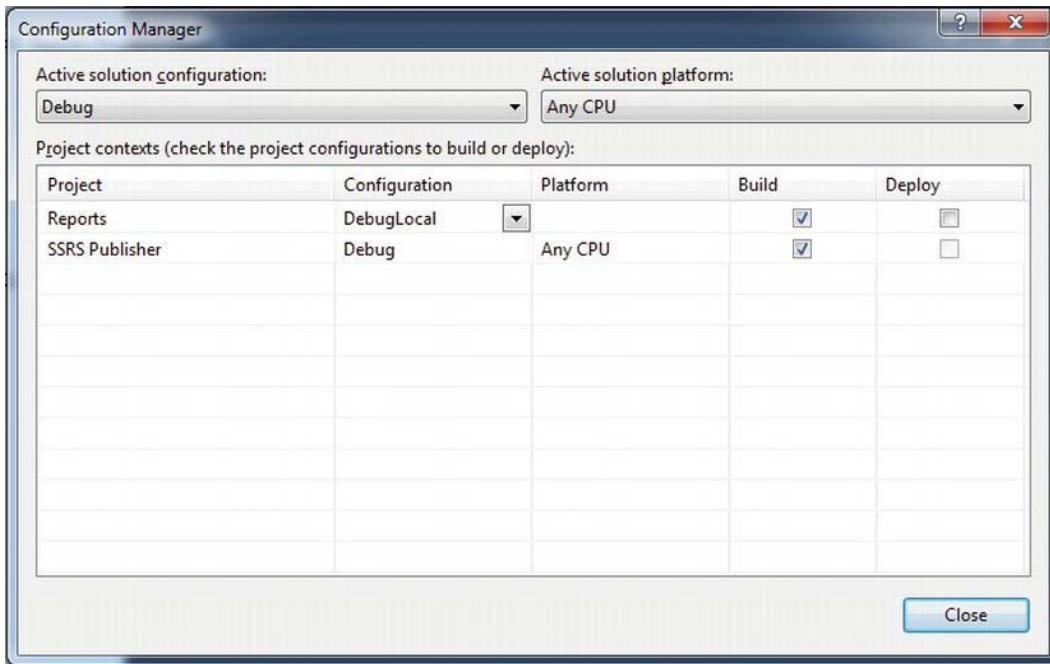


Figure 8-9. Visual Studio 2012 Configuration Manager

By default, you see the setup for the currently active configuration. You can select other configurations by choosing from the Active solution configuration drop-down list. As you can see, for each project in your solution, checkboxes are in the Build and the Deploy columns.

Each time you start the project, you want to build it so that you will always run the latest version of your report. If the Deploy checkbox is also checked, then whenever you start the project in that configuration, Visual Studio 2012 deploys the reports to the specified server that is setup in the report project properties. However, for most configurations, such as when you're debugging locally, you won't want to deploy your report to the server each time the project is started, so keeping the deploy checkbox unchecked is what you would want.

Deploying Reports Through the Solution Explorer

You can also manually deploy reports from the Solution Explorer once you are satisfied that your reports are ready to be deployed. The following is a list of options for deploying from the Solution Explorer. You deploy your reports by right-clicking each of the following items and selecting Deploy:

- The solution:** Deploys the reports and data sources in all the Report Server projects in your solution to the server that has been set up in each of the project's properties. This will deploy all projects in your solution, so use this only when you need to deploy everything.

- *The project:* Deploys the reports and data sources in the specific project in your solution to the server that has been set up in the specified project's properties. This option is only available for Report Server projects.
- *The report:* Deploys an individual report or data source from a project in your solution to the server that has been set up in the project properties containing the report you're deploying.

Figure 8-10 shows an example of how to deploy all the reports in a report project.

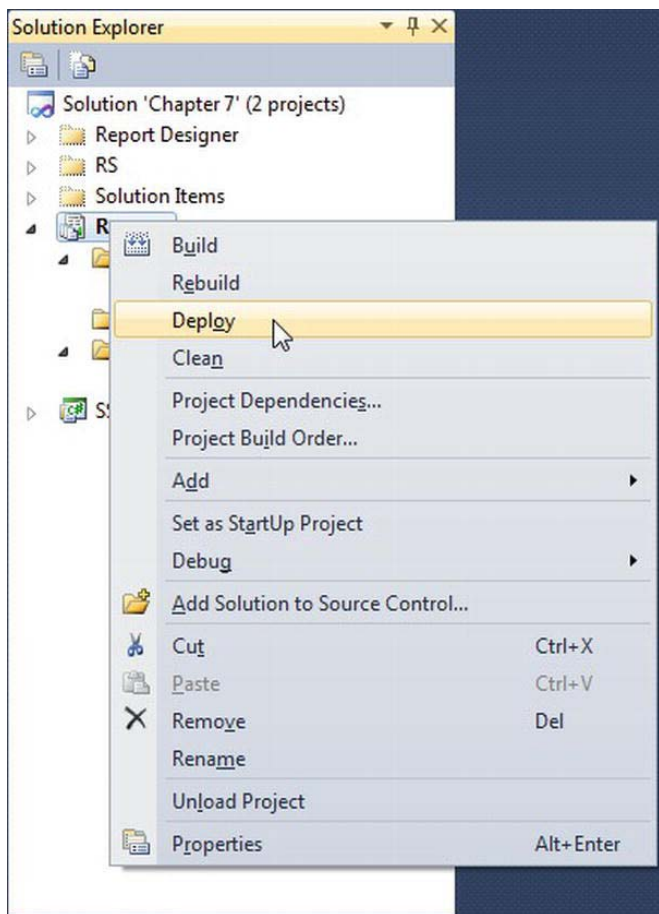


Figure 8-10. Deploying a solution through the Solution Explorer

Using the rs.exe Utility

In order to access one of the SSRS Web services, Reporting Services gives us a command-line utility called rs.exe. This utility uses script files written in VB .NET, the only language you can use with the rs utility, to access the Web service and allows us to call any of the published methods in the web service. In this example, you will create folders, deploy a report, and create a new data source through the rs.exe utility.

You can see the two files used in this example under the RS folder in the code examples for Chapter 8. You will find an RDL file containing our report definition that you will deploy and an RSS file containing the VB .NET code used to deploy the report and create the data source needed for the report. If you do not have your code in the folder C:\Pro_SSRS\CH8\RS, some configuration will be needed.

The RSS file has two global variables declared at the top of the file. These hold the information of where the report will be deployed on the report server and where the RDL files exist on your computer. You can see these two variables in Listing 8-1. You should make any necessary changes to these variables based on your setup.

Listing 8-1. RSS File Configuration

```
Dim parentPath As String = "Pro_SSRS/Chapter 8"
Dim reportPath As String = "C:\Pro_SSRS\CH8\RS"
```

Now that you have the RSS file configured, we will discuss the three Web service methods that are used. The first is CreateDataSource. This method creates a new data source on the report server and takes the following five arguments:

- **DataSource:** The name of the data source to be created
- **Parent:** The full path on the report server where the data source will be created
- **Overwrite:** A Boolean variable used to express whether to overwrite an existing data source of the same name
- **Definition:** An object of the type DataSourceDefinition that holds the connection properties of the data source
- **Properties:** An array of Property[] objects that hold property names and values for the data source

The second Web service method used is CreateReport. This method creates a new report on the server and takes the following five parameters:

- **Report:** The name of the report to be created
- **Parent:** The full path on the report server where the report will be deployed
- **Overwrite:** A Boolean variable to express whether to overwrite an existing report of the same name
- **Definition:** A Byte array holding the report definition
- **Properties:** An array of Property[] objects that hold property names and values for the report

The last Web service method used is CreateFolder. This method creates a new folder on the server and takes the following three parameters:

- Folder: The name of the folder to be created
- Parent: The parent name of the folder that you are creating
- Properties: An array of Property[] objects that hold property names and values for the new folder

The main function in the RSS file is calling three internal functions, one that utilizes the CreateFolder method, one that utilizes the CreateDataSource method, and one that utilizes the CreateReport method. You can use these three functions to create more folders and data sources or deploy more reports depending on your needs. In this example, we call all three functions once, but you can modify this file to deploy any number of reports or data sources that you need. You can see the full code for the RSS file in Listing 8-2.

Listing 8-2. RSS File Contents

```
Dim parentPath As String = "Pro_SSRS/Chapter 8"
Dim reportPath As String = "C:\PRO_SSRS\Chapter 8\RS\"

Public Sub Main()

    ' Initialize the reporting services credentials
    rs.Credentials = System.Net.CredentialCache.DefaultCredentials

    ' Pass our path information to create the folder structure
    CreateFolder(parentPath, "/")

    ' Pass our data source information to the create data source function
    CreateDataSource("Pro_SSRS", "SQL", "data source=(local);initial catalog=Pro_SSRS")

    ' Pass our report name to the deploy report function
    DeployReport("EmployeeServiceCost")

End Sub

Public Sub CreateDataSource(name As String, extension As String, connectionString As String)
    ' Define the data source definition.
    Dim dataSourceDefinition As New DataSourceDefinition()

    dataSourceDefinition.CredentialRetrieval = CredentialRetrievalEnum.Integrated
    dataSourceDefinition.ConnectionString = connectionString
    dataSourceDefinition.Enabled = True
    dataSourceDefinition.EnabledSpecified = True
    dataSourceDefinition.Extension = extension
    dataSourceDefinition.ImpersonateUser = False
    dataSourceDefinition.ImpersonateUserSpecified = True

    'Use the default prompt string.
    dataSourceDefinition.Prompt = Nothing
    dataSourceDefinition.WindowsCredentials = False
```

```

Try
    ' Create the datasource through the web service method
    rs.CreateDataSource(name, "/" + parentPath, False, dataSourceDefinition, Nothing)

    ' Display success message on creation
    Console.WriteLine("Data source {0} created successfully", name)
Catch e As Exception
    ' If the creation failed, catch exception and display the result
    Console.WriteLine(e.Message)
End Try

End Sub

Public Sub DeployReport(ByVal reportName As String)

    Dim definition As [Byte]() = Nothing
    Dim warnings As Warning() = Nothing

    Try
        ' Attempt to open the report as a file stream to read report definition information
        Dim stream As FileStream = File.OpenRead(reportPath + reportName + ".rdl")
        definition = New [Byte](stream.Length) {}
        stream.Read(definition, 0, CInt(stream.Length))
        stream.Close()

    Try
        ' Attempt to deploy the report via the web service
        warnings = rs.CreateReport(reportName, "/" + parentPath, False, definition,
Nothing)

        If Not (warnings Is Nothing) Then
            Dim warning As Warning
            For Each warning In warnings
                Console.WriteLine(warning.Message)
            Next warning

        Else
            Console.WriteLine("Report: {0} published successfully with no warnings",
reportName)
        End If

    Catch e As Exception
        Console.WriteLine(e.Message)
    End Try

    Catch e As IOException
        Console.WriteLine(e.Message)
    End Try

End Sub

```

```

Public Sub CreateFolder(ByVal folderName As String, ByVal ParentFolder As String)
    Dim extraName As String
    Dim newFolder As String

    if( folderName.IndexOf("/") <> -1 )
        newFolder = folderName.Substring(0, folderName.IndexOf("/"))
        extraName = folderName.Substring(folderName.IndexOf("/") + 1)
    else
        newFolder = folderName
        extraName = ""
    end if

    Dim props(0) As [Property]
    Dim folderProp As new [Property]()

    folderProp.Name = "Folder Name"
    folderProp.Value = newFolder

    props(0) = folderProp

    rs.CreateFolder(newFolder, parentFolder, props)
    Console.WriteLine("Folder " + newFolder + " created")

    if( extraName <> "" and parentFolder = "/" )
        CreateFolder(extraName, parentFolder + newFolder)
    else if( extraName <> "" )
        CreateFolder(extraName, parentFolder + "/" + newFolder)
    end if
End Sub

```

■ **Note** You can deploy a set of data sources and reports to multiple SSRS servers easily with the rs.exe utility. To mirror deployment with the same RSS script file, simply rerun the RS command with a different server parameter. Creating a batch file that calls rs.exe multiple times with different server destinations is an easy way to deploy projects exactly the same way to several locations.

The only function that gets called more than once is our user defined CreateFolder method. This is a recursive function that will parse the name of the folder you want to create and build it piece by piece. In order to cut down on space, there is no error checking here. So, if you run this example more than once, you will need to clean up the folders that this creates or it will throw an exception due to the fact that the folder already exists.

From a command prompt, move to the directory where the RDL and RSS file are being stored. Type the following command in the command prompt, and press Enter to run it. Change the report server URL to reflect your report server.

```
rs -i PublishReports.rss -s http://localhost/reportserver
```

You can see the command and the results in Figure 8-11.

You now have a new data source and report deployed to your report server. The RSS file included in the example code can be used as a starting point to script your own custom report project deployments.

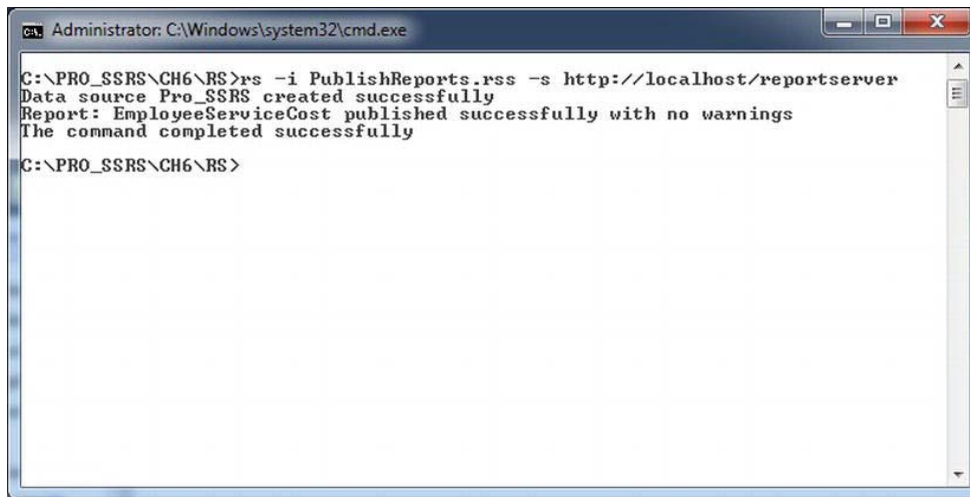


Figure 8-11. Deploying a report and data source with the rs.exe utility

Using the Report Server Web Service

The method provided by SSRS 2012 for deploying reports programmatically is to use one of the Report Server Web services using the SOAP API (which we will use again in Chapter 9 to write a report viewer). In this section, we'll look at deploying reports to SSRS 2012 by using a Windows Forms application that simulates what customers need to do after they have an RDL file ready for deployment:

- Select a report server to publish their report to.
- Select from a displayed list of folders on that server to determine which folder on the server to publish the report to.
- Browse to the RDL file that is to be uploaded from the local machine to their report server.

There are several web service endpoints available to you when dealing with SSRS 2012. The old versions of each web service are still there for backward compatibility and have the ability to manage either native or SharePoint mode installs separately. These are deprecated and it is advisable to upgrade to the newest version, which allows you to control both types of installations from one web service. We will be using the newest management web service in this example.

You will use the Report Server Web service to get a list of folders on the server and then upload the report to the server. In this example, you'll upload some of the reports created for the health-care provider. In the health-care setting, it's important to maintain strict control over the report folders and their permissions on the server, so you won't allow the users to create a new folder; they will only be allowed to upload to the existing folders that they already have permission to use.

The `CreateCatalogItem` method of the Report Server Web service allows you to deploy the report to the report server by creating a copy of the report on the server from an RDL file that you provide. This is the same method as described in the previous section, "Using the rs.exe Utility."

■ **Note** The `CreateReportEditSession` method might pass sensitive data, including user credentials, over the network. You should use SSL encryption whenever possible when making Web service calls.

First, you'll create a new C# Windows Forms solution with Visual Studio 2008. Call this project `SSRS_Publisher`. This example will be in C#, but the source examples from the website will also have a working version written in VB.NET as well. Once your new project is created, rename the `Form1.cs` file to `Publisher.cs` and allow VS to update all of the references.

Accessing the Web Service

You need to add a reference to the SSRS 2012 Report Server Web service, which is the same as you did in Chapter 5 for the report viewer. You do this by selecting **Project > Add Web Reference**, right-clicking the references in the Solution Explorer and selecting **Add Web Reference**; alternatively, you can right-click the project in the Solution Explorer and select **Add Web Reference**. Depending on the version of VS and .NET you are working with, you may not immediately see the option to add a web reference.

If you don't see the option for **Add Web Reference** in those locations, you may need to use a different method to add the reference. You will need to select **Project > Add Service Reference**. From this screen, you will click the advanced button on the bottom left of the window. In the advanced settings window, you will click on the **Add Web Reference** button in the bottom left of the screen to bring up the web reference window.

When the dialog box appears, enter the following URL:

`http://localhost/reportserver/reportservice2010.asmx`

Substitute the name of your server for `localhost` in the preceding URL if you are using a remote server and then click the arrow button next to the URL. You may be prompted for your credentials at this point to verify that you have permissions to access the web service. You should now see a dialog box similar to the one in Figure 8-12.

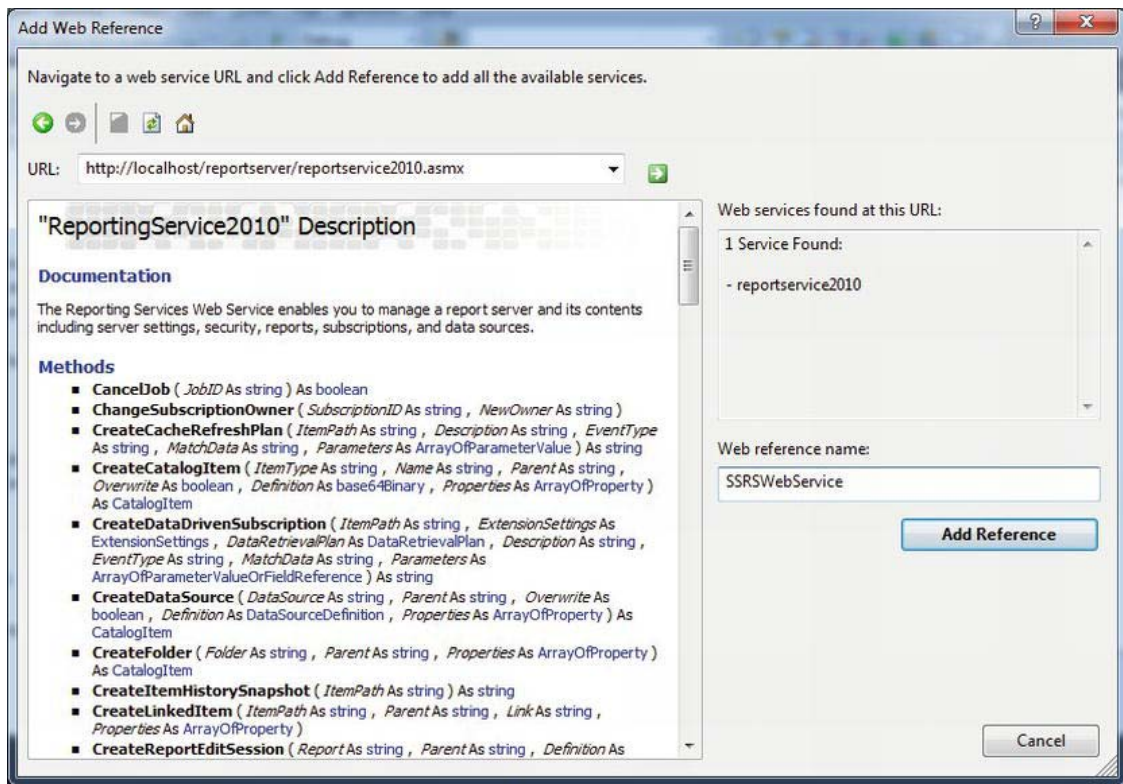


Figure 8-12. Add Web Reference dialog box

In the Web reference name textbox, enter **SSRSWebService**. This name is how the Web service is referenced in the code. After this dialog box is closed and the reference is added to the project, add the following using directives to the code at the top of your publisher.cs code with the rest of the standard using directives:

```
using SSRS_Publisher.SSRSWebService;
```

Now you can reference the Web service much more easily because you don't have to enter the fully qualified namespace. You'll also add three others using directives below this one that will allow you to access the Web services and I/O-specific functions more easily in your code, as shown in Listing 8-3. You can see what all of this together should look like in Figure 8-13.

Listing 8-3. Using Directives

```
using System.IO;
using System.Text.RegularExpressions;
using System.Web.Services.Protocols;
```

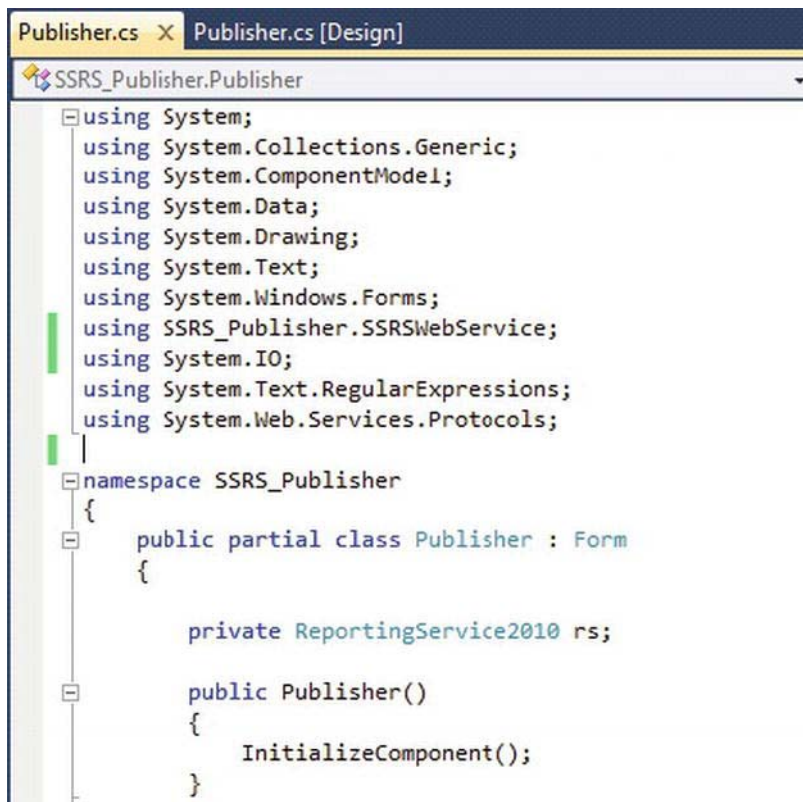


Figure 8-13. Adding directives to your publisher project

Laying Out the Form

From the Toolbox, add one label, one textbox, and one button near the top of the form, set the label text to **Report server**, and name the textbox reportServer in the textbox properties. Name the button getFolders and set its text to the word **Go**. You'll use the reportServer textbox to allow users to enter in the report server name that they want to deploy to.

Next, add a TreeView control to the center of the form and name it ssrsFolders. You'll use it to display a list of available folders on the server where you can upload your report when the user enters in a server name and clicks the Go button.

Add one label, one textbox, and one button near the bottom of the form; set the label text to **Report file**; name the textbox reportFile; name the button browseFile; and set the text of the button to **Browse**. You'll use the reportFile textbox to accept or display the path name and file name of the report file you want to upload to the specified SSRS 2012 server. You'll use the Browse buttons to choose the form that accepts the users' input and allows them to select a report file for deploying.

After you've added all these controls, the form should look something like Figure 8-14.

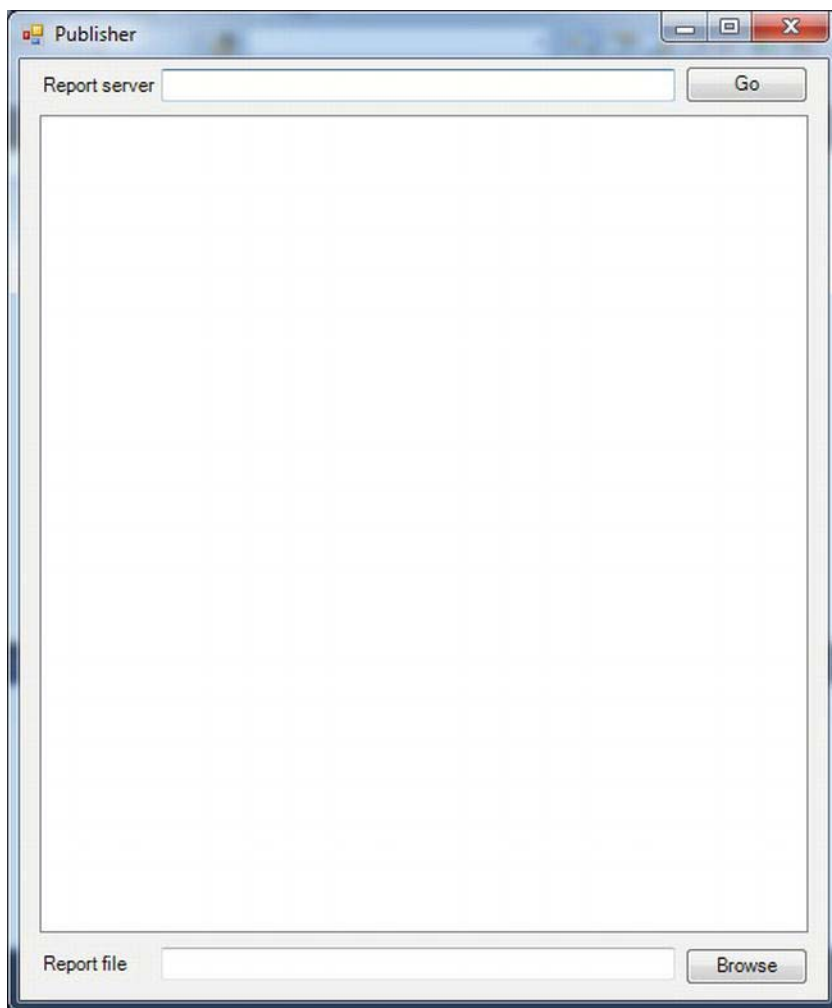


Figure 8-14. Publisher, the example report publisher

Coding the Form

Now that you have the form laid out, you'll add code to handle the functions necessary to allow users to do the following:

- Enter in a server name.
- Get a list of the folders available on that server.
- Select a target folder and an RDL file to upload.

Allowing Users to Enter a Server Name

When a user types in a server name on the form, the code needs to build the URL that fully identifies the SSRS 2012 server to deploy the RDL file to.

First, you will create a class-level variable to hold the reference to the report server, much as you did in the SSRS viewers in Chapter 5. You do this by adding the variable definition `private ReportingService2010 rs;` to the class, as is shown in Listing 8-4.

Listing 8-4. Class-level Variable in Context

```
public partial class Publisher : Form {
private ReportingService2010 rs;
public Publisher()
```

To build the URL that identifies the server based on the user input, you first instantiate the Report Server Web service and then set the URL property to reflect the name of the server that the user entered into the `reportServer` textbox. We will want to form this URL for the user, so they only need to enter the server name to connect.

To do this, first create a short function to check the server name that the user enters and then append the rest of the path name to the Reporting Services Web service to make up the complete URL necessary to reference the Reporting Services Web service on the desired server. By constructing the URL based on the user's input, you can use the report deployment application to deploy the reports on any SSRS 2012 server where the user has permission to do so. Start by adding the code shown in Listing 8-5 into the `Publisher` class.

Listing 8-5. Get Report Server URL

```
private string GetRSURL() {
    if (reportServer.Text.StartsWith("Error! Hyperlink reference not valid."))
        return reportServer.Text + "/reportserver/ReportService2010.asmx";
    else
        return "Error! Hyperlink reference not valid." + reportServer.Text +
            "/reportserver/ReportService2010.asmx";
}
```

Populating the TreeView Control With a List of Folders

Now you'll use the Reporting Services Web service to retrieve a list of objects from the server and use them to populate the `ssrsFolders` TreeView control. We'll do this by placing a call to the `ListChildren` method of the Report Server Web service. The `ListChildren` method takes two parameters:

- **Item:** The full path name of the parent folder.
- **Recursive:** A Boolean expression that indicates whether to return the entire tree of child items below the specified item. The default value is false.

■ **Note** The `ListChildren` method returns all objects on the report server, including data sources, report parts, and reports, not just folders. In this example, you'll filter out everything but the folders because you're only interested in showing the user the folder structure. You do this by using the `ItemType` property of the child object and then testing it against the string "Folder".

Make sure the `Publisher.cs` form is open in the design view and double-click the Go button. This creates an empty method to handle the button's click event. Add the code in Listing 8-6 to the method.

Listing 8-6. *Code to Populate the TreeView Control*

```
ssrsFolders.Nodes.Clear();
rs = new ReportingService2010();
rs.Credentials = System.Net.CredentialCache.DefaultCredentials;
CatalogItem[] items = null;
rs.Url = GetRSURL();

TreeNode root = new TreeNode();
root.Text = "Root";
ssrsFolders.Nodes.Add(root);
ssrsFolders.SelectedNode = ssrsFolders.TopNode;

// Retrieve a list items from the server
try
{
    items = rs.ListChildren("/", true);
    int j = 1;

    // Iterate through the list of items and find all of the folders and display them to the user
    foreach (CatalogItem ci in items)
    {
        if (ci.TypeName == "Folder")
        {
            Regex rx = new Regex("/");
            int matchCnt = rx.Matches(ci.Path).Count;
            if (matchCnt > j)
            {
                ssrsFolders.SelectedNode = ssrsFolders.SelectedNode.LastNode;
                j = matchCnt;
            }
            else if (matchCnt < j)
            {
                ssrsFolders.SelectedNode = ssrsFolders.SelectedNode.Parent;
                j = matchCnt;
            }
            AddNode(ci.Name);
        }
    }
}
```

```

}

catch (SoapException ex)
{
    MessageBox.Show(ex.Detail.InnerXml.ToString());
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

// Make sure the user can see that the root folder is selected by default
ssrsFolders.HideSelection = false;

```

Right below the method for the Go button's click event, add the following method as shown in Listing 8-7.

Listing 8-7. AddNode Method

```

private void AddNode(string name) {
    TreeNode newNode = new TreeNode(name);
    ssrsFolders.SelectedNode.Nodes.Add(newNode);
}

```

You tell the ListChildren method to start at the root folder by passing in a "/" as the starting point, and also set the recursive option to true, which causes the ListChildren method to iterate through all the folders and subfolders on the SSRS 2008 server. You use this information to create nodes in the TreeView control to display each folder in a hierarchy that represents the hierarchy of the folders on the server. Use a regular expression to look for the number of "/" characters in the path of each CatalogItem to determine how deep you are in the hierarchy (one level, two levels, and so on).

Opening the RDL File and Uploading It to the Server

Now you need to add some code to allow users to browse for the file that they want to upload. You'll want to limit the users to browse for files ending in "rdl" by default, because this is the native extension for SSRS 2008 report definition files. You also want to read the selected node in the ssrsFolders TreeView so that you know what folder users have selected to deploy the report to on the SSRS 2008 server.

Start by reading the path of the selected node from the TreeView control and turning it into a path name you can use with SSRS 2012's CreateCatalogItem method.

Make sure the Publisher.cs form is open in design view and double-click the Browse button. This creates an empty method to handle the button's click event. Add the code in Listing 8-8 to the method.

Listing 8-8. Code to Browse for an RDL File

```

private void browseFile_Click(object sender, EventArgs e)
{
    // Get the full pathname from the treeview control
    string pathName = ssrsFolders.SelectedNode.FullPath;

    if (pathName == "Root")
        pathName = "/";
    else

```

```

{
    // Strip off the Root name from the path and correct the path separators for
    use with SRS
    pathName = pathName.Substring(4, pathName.Length - 4);
    pathName = pathName.Replace(@"\", "/");
}

byte[] definition = null;
Warning[] warnings = null;
string warningMsg = String.Empty;

OpenFileDialog openFileDialog = new OpenFileDialog();
openFileDialog.Filter = "RDL files (*.rdl)|*.rdl|All files (*.*)|*.*";
openFileDialog.FilterIndex = 1;
if (openFileDialog.ShowDialog() == DialogResult.OK)
{
    try
    {
        // Read the file and put it into a byte array to pass to SRS
        FileStream stream = File.OpenRead(openFileDialog.FileName);
        definition = new byte[stream.Length];
        stream.Read(definition, 0, (int)(stream.Length));
        stream.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

    // We are going to use the name of the rdl file as the name of our report
    string reportName =
    Path.GetFileNameWithoutExtension(openFileDialog.FileName);
    reportFile.Text = reportName;

    // Now lets use this information to publish the report
    try
    {
        rs.CreateCatalogItem("Report", reportName, pathName, true, definition,
        null, out warnings);

        if (warnings != null)
        {
            foreach (Warning warning in warnings)
            {
                warningMsg += warning.Message + "\n";
            }
            MessageBox.Show("Report creation failed with the following
            warnings:\n" + warningMsg);
        }
        else
            MessageBox.Show(String.Format("Report: {0} created
            successfully with no warnings", reportName));
    }
}

```



```

    }
    catch (SoapException ex)
    {
        MessageBox.Show(ex.Detail.InnerXml.ToString());
    }
}

```

The code starts by getting the full path on the SSRS 2012 server where the user has selected to place the report from the `ssrsFolders` `TreeView` control, strips the word *Root* off the front of the path, and then replaces all occurrences of a backslash in the string with the forward slash needed for SSRS 2012.

You then set the options for the `openFileDialog` control so that it browses by default for files with the RDL extension and then displays the dialog box to the user. If the user makes a selection, then the file is opened using a `FileStream` object and read from the stream into a byte array. That's because the SSRS 2012 `CreateCatalogItem` method expects the contents of the RDL file to be passed in as a byte array.

Next, the file name that the user selected is read and used as the title for the report in SSRS 2012. After you have a title, you have everything necessary to upload the report, which you do by calling the `CreateCatalogItem` method with the values you've created. You can see at the end of Listing 8-8 where the necessary code has been added to the complete listing.

Running the Application

Now let's run the example. Start the project and, when the form displays, enter the name of your report server in the `Server` textbox and click `Go`. `Localhost` was used in the example; use the name of your server if it is different. Your form now looks similar to Figure 8-14, with the folders on your SSRS 2012 server displayed and the `Root` folder highlighted.

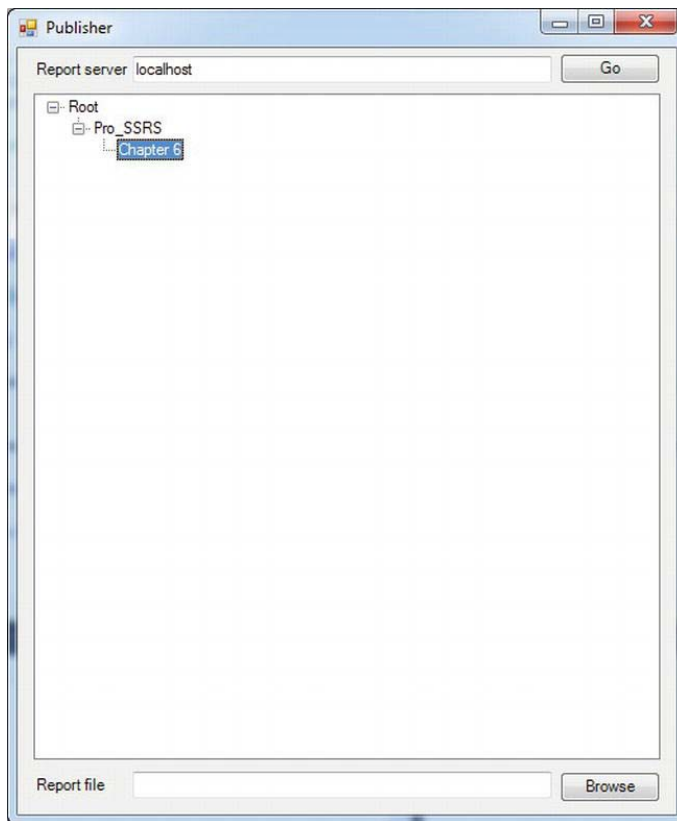


Figure 8-15. Complete report publisher showing folders on SSRS server

■ **Note** You need to deploy the shared data source from the Chapter 8 sample code prior to uploading the sample report included in Chapter 8. The report uses this shared data source and will tell you that it could not publish the report because it can't find the shared data source on the SSRS 2012 server. Also note that the shared data source in this chapter is in the local Chapter 8 folder instead of the Pro_SSRS folder we have been using. We did this because, when uploading the report using the SSRS Publisher, it will make the connection to the shared data source Pro_SSRS only if it resides in the same folder as you deploy the report to.

By default, the Root folder on your server has been selected. Drill down and select the Pro_SSRS node in the TreeView. After you've selected the folder to use, click Open, and select the report in the EmployeeServiceCost.rdl file included in the Chapter 8 sample code. When you click Open, your report publishing application uses the Report Server Web services CreateCatalogItem method and publishes the report to the selected server.

In this case, the Pro_SSRS folder on the localhost server has been selected, and we're uploading a report called EmployeeServiceCost.rdl. If you navigate to that folder with your Web browser, you'll see something similar to the screen shown in Figure 8-16.

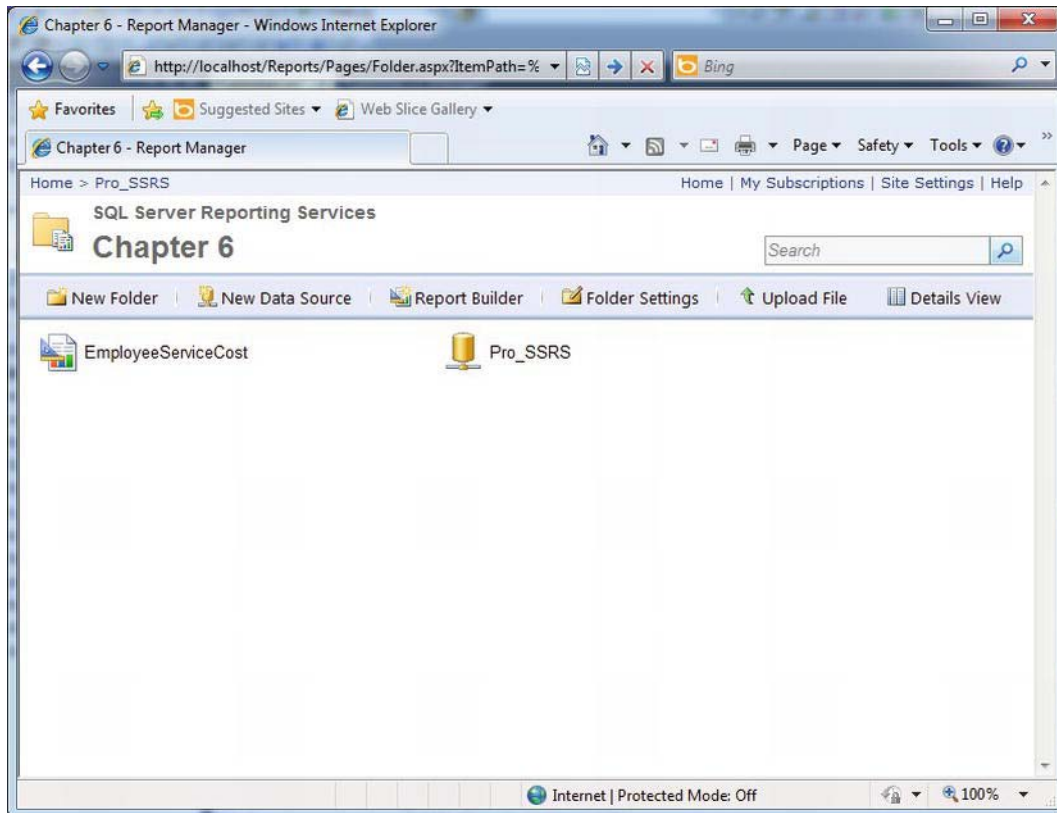


Figure 8-16. Report Manager showing uploaded report

You now have a Windows Forms application that allows files to be uploaded to your report server. This can be a handy way to add the capability to upload or update reports from within an application without needing the user to interact with the SSRS 2012 Report Manager directly.

This can be especially useful if you want users to interact with all aspects of SSRS 2010 from within your application. In Chapter 5, you developed an application that allows users to display reports from within the application. By combining that application with this report publisher, you can handle a number of non-administrative tasks directly from your application. You also could expand on this example by providing the user with some additional options:

- You could allow the user to create shared data sources.
- You could allow the user to enter the name of the report to publish instead of taking the name from the RDL file itself. For example, you could add another textbox to the form and read it for the report title. So, instead of the following code:

```
// We are going to use the name of the RDL file // as the name of our report string
reportName =
Path.GetFileNameWithoutExtension(openFileDialog.FileName);
```

you could do the following:

```
// We are going to read the contents of textbox reportTitle
// as the name of our report
string reportName = reportTitle.text
```

- You could allow the user to set other properties of the report, such as adding a description that will be displayed in Report Manager in the detailed list view. For example, you could add a Description property:

```
Property[] itemProps = new Property[1];
Property itemProp = new Property();
itemProp.Name = "Description";
itemProp.Value = "Employee Service Cost by Patient";
itemProps[0] = itemProp;
```

- You could then change your call to CreateReport from the following code:

```
rs.CreateCatalogItem("Report", reportName, pathName, true, definition, null, out warnings);
```

to the following code instead:

```
rs.CreateCatalogItem("Report", reportName, pathName, true, definition, itemProps, out
warnings);
```

With Report Server Web service, there isn't much you can't do. For more details on additional properties that you can set on reports or to find out more about the SOAP API, see the SQL Server 2012 Books Online. We will be diving into the web service even more in the next chapter.

Summary

In this chapter, we discussed and demonstrated how to use the Report Builder 3.0 utility, the rs.exe application, and BIDS/Visual Studio to deploy reports to your SSRS 2012 server. Additionally, you used the SOAP APIs through the Report Server Web service to list the folders on the selected server and then allowed the user to select an RDL file and upload it to the user-specified folder on the selected report server using both a Visual Studio C# windows application and the rs.exe utility. We also examined some of the additional features you could provide the application by using a few of the many methods and properties exposed by the SSRS 2012 Web service.

Rendering Reports from .NET Applications

Report rendering is the process of outputting the results of a report into a specific format. You pass the appropriate parameters to SSRS, telling it what report you want to run and optionally what format you want the output in, any user credentials, and the actual report parameters. SSRS 2012 then renders the report and returns the results.

The manner in which you pass these parameters and how the results are returned depends on the SSRS 2012 method you're using to render the report. Once SSRS has the information it needs, based on the particular report you're running, it queries the appropriate data sources. SSRS 2012 uses the passed credentials and parameters if appropriate, renders the report into an intermediate format, and then renders and filters this intermediate format into the final display format requested.

With SSRS 2012, you can render reports in three ways from a .NET application:

Using a URL: You can build a URL that allows the client to access the report on the report server and supply any appropriate parameters, including rendering format, login information, report criteria, and report filters. This method is one of the most flexible because it will work with almost any language and platform that can host or use a Web browser or a Web browser control. In fact, it works with any language that can create a properly formatted URL and that can launch a Web browser using that URL. We can use the built-in WebBrowser control in .NET with formatted URLs to render a report.

Using the SOAP API: You can use the SOAP API, also known as the Report Server Web service, to render the report. This returns the rendered data as a stream that you then display. This is a more difficult method of rendering because the information you get back from the server is essentially a binary stream of data, and you don't have the benefit of having the server-browser combination do the actual work of displaying the data. However, you can use the Report Server Web service for more than just rendering: you can use it to access the report server's complete functionality. We'll show how you can use the Report Server Web service in your solution to provide information about the reports you're rendering, such as the report parameters that the reports use.

Using the Report Viewer control3.0: You can use the Report Viewer control. This option for SSRS 2012 provides a simple-to-use Windows Forms application or ASP.NET user control that allows you to simply drag and drop the control onto a Windows or Web Form, set some properties, and render a report. One of the main benefits of the Report Viewer control is that it allows you to render reports that are on an SSRS 2012 report server as well as render reports locally, without the need for a report server.

■ **Note** The Report Viewer controls are included with current versions of SQL Server and Visual Studio. You can also download the redistributable package for machines that will be running the application and that do not have Visual Studio/SQL Server installed.

While the Report Viewer controls are the method that most Visual Studio users will probably employ to render reports, the most universally usable rendering method is via URL access. In this case, SSRS 2012 provides some defaults for most reporting options. For example, SSRS provides a default user interface for entering parameter and filter information. It prompts you for login information if necessary, and it defaults to rendering in HTML format. You get all this simply by passing the URL of the report from the browser. This is most useful if you're rendering your reports using just your Web browser, with no other controlling application.

You can optionally pass parameters along with the URL to change these default behaviors, provide login information, change the rendering format, hide the parameter toolbar, and much more. This is useful if you have a custom application and want to control these options yourself, rather than provide the default user interface.

You can perform many of the same actions using the Report Server Web service, but there are no real defaults and the actual display of the returned data to the user is left up to the application developer. By using the Web browser in an ASP.NET application, or embedded in a Windows Forms application, you get the benefits of the URL rendering method, but you can exercise control over it. This provides users with a more integrated experience. The methods covered in this chapter apply largely to both Windows Forms and Web Forms applications. For one of the projects, we'll show how to build a Windows Forms-based report viewer that allows you to use SSRS 2012 as the reporting solution for your application.

In this chapter, you'll do the following:

URL rendering access: Learn how to build a URL through which a client application can access a report on the report server. You'll use the Employee Service Cost report in the example (available with the code download for this chapter in the Source Code/Download section of the Apress Web site at www.apress.com).

URL reporting parameters: Explore the reporting parameters that, when specified in the URL, control how the report is rendered. You can specify the actual format (for example, HTML or PDF). You can specify that a specific page in a report be rendered, or you can search for a particular word and start rendering on that page.

URL viewer application: Build a simple .NET Windows Forms application that accesses and renders a report via a WebBrowser control that you'll embed in a form.

Report Server Web service calls: Use calls to the Report Server Web service to query for the report criteria and filter parameters. This allows you to display these on your Windows Forms application. You'll then use the SOAP API again to see whether the parameters have a list of values from which the user can select. If so, you'll use those values to populate combo boxes for each parameter. You'll also give the user a combo box to select a rendering format. You'll then display all the selections to the users and use their selected values to

create a URL that contains all the information necessary to run the report. Finally, you'll use this URL with the embedded Web browser to render the report with user-entered report parameters and rendering commands.

Report Viewer control, server-side mode: Use the Report Viewer control to render the same report using the control's server-side mode.

Report Viewer control, local rendering: Use the Report Viewer control and a locally populated dataset to render a report locally without an SSRS report server.

Report Viewer control in an ASP.NET application: Use the Report Viewer control and render a server-side report. We will make use of the Web service to retrieve a list of reports to choose from.

Before you run the included examples, make sure to read the ReadMe.htm. It is located in a file in the samples root folder. If you have the code open in Visual Studio, it will be under the Solution Items folder. It contains setup and configuration steps that are required before running the examples.

Implementing URL Access

In this section, we'll show you how to build a URL that accesses the desired reports on the report server and passes the appropriate parameters to the report.

The syntax for the entire URL breaks down into two parts. The first part specifies the path to the report file, and the second specifies the parameters. The full URL syntax is as follows:

```
http://server/virtualroot?[/pathinfo]&prefix:param=value [&prefix:param=value]...n]
```

The syntax for an SSRS installation in SharePoint integration mode will resemble:

```
http://server/subSiteName/_vti_bin/reportserver?[/pathinfo]&prefix:param=value
[&prefix:param=value]...n]
```

Table 9-1 describes each component of the URL.

Table 9-1. URL Access Parameters

Parameter	Description	Supported Values
server	Specifies the name of the SSRS 2012 Web server	None
subSiteName	The sub site of SharePoint that contains the integrated SSRS installation	None
virtualroot	Specifies the virtual root of the SSRS 2008 Web service	None
?	Separates the application virtual root from the parameters	None
[/pathinfo]	Specifies the optional path to a folder containing the report	None
&	Separates individual parameters	None

Parameter	Description	Supported Values
Prefix	When used, indicates that the following parameter is a command to the server itself versus a report parameter	rc: Rendering control rs: Report server command dsu: User dsp: Password
Param	Specifies the command parameter name when used with a prefix; otherwise, specifies a report parameter name	None
Value	Specifies the value of the parameter	None

We'll now walk you through the steps of building a URL to access the Employee Service Cost report.

URL Report Access Path Format

As indicated in the previous section, the path to access the appropriate report starts with the name of the report server itself; in this case, we will use `http://localhost` and use an SSRS installation that is in native mode. Following that is the name of the SSRS 2012 virtual root folder (such as `/reportserver`, which is the default folder during installation).

■ **Note** The `/reports` virtual root folder is mapped to the Report Manager application that ships with SSRS 2012. If you navigate to this URL, you'll find that it lists the folders and the reports within the folders that are on your report server through Report Manager.

You then add `?` to the path to let SSRS 2012 know that everything following the URL is a parameter. Next is the optional path information, where you can specify any subfolder within the base folder that you use to organize your reports, such as `/Pro_SSRS/Chapter_9`. Finally, you have the name of the actual report, such as `EmployeeServiceCost`.

So, in this example, the full path to access the Employee Service Cost report is as follows:

`http://localhost/reportserver?/Pro_SSRS/Chapter_9/EmployeeServiceCost`

We'll now cover the section of the URL where you specify any necessary parameters.

URL Parameters and Prefixes

You now need to pass the appropriate parameters to the report. You're interested in several categories of parameters:

Report parameters (no prefix): These parameters are supplied to the report's underlying queries and are used as filters for the information as it's rendered. Thus, they control exactly which data the report displays.

HTML Viewer parameters (rc:): These parameters control which features of the Web-based report viewer are active and at which page the report viewer starts displaying the report. For example, you can use the FindString parameter to have the viewer start displaying a report on the first page on which a specific word is found.

Report Server command parameters (rs:): These parameters control the type of request being made and the format of the returned report. For example, we'll show how to use the rs:Command=Render&rs:Format=HTML4.0 parameter to render your report in HTML format in your report viewer.

Report Viewer Web Part command parameters (rv:): These parameters control the SharePoint integrated web part. These are very similar to the html viewer parameters since they will control the portal where you view reports on a SharePoint site. For example, you can use the AsyncRender parameter to control whether the report is rendered asynchronously or synchronously. Web parts can also accept the rs:ParameterLanguage parameter in addition to the rv: group.

Report Parameters

Report parameters are the actual parameters passed to the underlying report, instead of instructions being sent to the report server. That is, you use them to pass criteria to your report, such as start and end dates, employee IDs, and so on. You can pass these parameters to the report's query, and they're created when you specify parameters for the data source. You can also use parameters as variables in the report and as values for filters.

HTML Viewer Commands

You use HTML Viewer commands to tell SSRS 2012 how to render the report. You can use the commands in Table 9-2 to control how the viewer appears to the user, as well as to control certain aspects of how the report appears in the viewer.

Table 9-2. *HTML Viewer Commands*

Parameter	Description	Default
Toolbar	Shows or hides the toolbar.	true
Parameters	Shows or hides the parameters area of the toolbar.	true
DocMap	Shows or hides the report document map. For information on what a document map is, see Chapter 6.	true
Zoom	Sets the zoom value. You can use a number representing a percentage or a string with standard values such as Page Width and Whole Page.	100
Section	Specifies the page number of the report to display.	1

Parameter	Description	Default
FindString	Specifies the text to search for in the report.	Not applicable
StartFind	Specifies the page number to start the search on, specified by FindString.	Last page of the report
EndFind	Specifies the page number to end the search on, specified by FindString.	Current page
FallbackPage	Specifies the page number to display if a FindString or DocMapID fails.	Not applicable
GetImage	Gets a particular icon for the HTML Viewer interface.	Not applicable
Icon	Gets the icon for a rendering extension. Not	applicable
Stylesheet	Specifies a style sheet to be applied to the HTML Viewer.	Not applicable
Device Information Setting	Specifies device information with the form rc:tag=value. The tag is the name of a device setting specific to the rendering extension that is currently being used.	Not applicable

Report Server Command Parameters

Report server command parameters, which are prefixed with rs:, tell the report server the type of request being made (see Table 9-3). You use them to retrieve report and data source information in XML and HTML format and to retrieve child elements such as the current folder's report names. You also use command parameters to tell the server you want to render a report and in what format and whether you want to render that report based on a preexisting snapshot.

Table 9-3. *URL Command Parameters*

Parameter	Description	Supported Values
Command	Specifies the type of request being made	ListChildren Render GetSharedDatasetDefinition GetDataSourceContents GetResourceContents GetComponentDefinition
Format	Specifies the format to render the report in	ATOM HTML4.0 MHTML IMAGE EXCEL CSV PDF XML WORD
Snapshot	Renders a report based on a snapshot	Valid ID of a snapshot
ParameterLanguage	Provides a language for passed parameters Browser	language
PersistStreamsstram	Renders a report as a single persistent stream.	True/False
GetNextStream	Gets the next data chunk from a report rendered with a persistent stream. This parameter replaces PersisStreamsstram in a URL when you want to receive the next chunk.	True/False
SessionID	Specifies a previously established session between the client and the report server.	Session Identifier
ClearSession	Clears a report from a report session when given a value of true. Will clear all instances of the report from the session.	True/False
ResetSession	Clears all reports from a session when given a value of true.	True/False
ShowHideToggle	Toggles the show/hide value of a section of the report	Integer value of the section

Credential Parameters

If, in previous versions of Reporting Services, you were using credential parameters to pass datasource connection information, you will have to modify these reports to upgrade them to a SSRS 2012 server. The credential parameters that passed username and password information to a report server for rendering were stored in plain text in the browser cache. This was very insecure and this set of parameters has been removed from SSRS 2012.

Report Viewer Web Part Commands

Report web part viewer commands are prefixed with `rv:` and are used to target the web part used to view reports in SharePoint mode. You can use these parameters to control how the report and its controls are displayed from the web part. You can also use these parameters to control how the report will be rendered. You can see these parameters in Table 9.4.

Table 9-4. Web Part Commands

Parameter	Description	Supported Values
Toolbar	Controls how the toolbar is displayed on the report viewer web part	Full Navigation None
HeaderArea	Controls how the header is displayed on the report viewer web part	Full BreadCrumbsOnly None
DocMapAreaWidth	Controls the pixel width of the document map	Integer
AsynchRender	Controls whether the report is rendered synchronously or asynchronously. Using a true value will render asynchronously.	Boolean
ParamMode Contr	ols how the parameter prompt is displayed in the report viewer web part	Full Collapsed Hidden
DocMapMode	Controls how the document map is displayed in the report viewer web part	Full Collapsed Hidden
DockToolBar	Controls where the web part toolbar is docked on the page	Top Bottom
ToolBarItemsDisplayMode Contr	ols which items are displayed in the web part's toolbar. This is a bitwise enumeration value of each of the toolbar items. To show all items, use the value -1.	Integer from (-1) to (2047)

Example URLs

Now that you've examined every component part of the URL, it's useful to look at a few complete sample URLs. You will need to have the report setup correctly from the `readme.html` file included from the chapter download examples. The following URL renders the report in HTML 4 and hides the HTML Viewer toolbar by setting the `rc:Toolbar` parameter value to `false` while also setting the Service Year parameter to 2009:

```
http://localhost/reportserver?/Pro_SSRS/Chapter_9/EmployeeServiceCost&rs:Command=Render&rs:Format=HTML4.0&rc:Toolbar=false&ServiceYear=2009
```

The next example passes a report parameter for ServiceYear of 2009 and hides the input display of user-supplied parameters:

```
http://localhost/reportserver?/Pro_SSRS/Chapter_9/EmployeeServiceCost&rs:Command=Render&rs:Format=HTML4.0&rc:Parameters=false&ServiceYear=2009
```

The next one uses the `rs:Format` parameter to set the default output format to PDF for the service year 2009:

```
http://localhost/reportserver?/Pro_SSRS/Chapter_9/EmployeeServiceCost&rs:Command=Render&rs:Format=PDF&serviceyear=2009
```

This example hides the report document map:

```
http://localhost/reportserver?/Pro_SSRS/Chapter_9/EmployeeServiceCost&rs:Command=Render&rc:DocMap=false&ServiceYear=2009
```

This final example – in SharePoint integrated mode which you may not have set up – will hide the toolbar:

```
http://localhost/_layouts/ReportServer/RSViewerPage.aspx?rv:RelativeReportUrl=/Chapter_9/EmployeeServiceCost.rdl&rv:Toolbar=None
```

You've taken a brief look at the commands you need to render a report through a URL in both native and SharePoint mode. In the remainder of this chapter, we'll show how to create your own report viewer as well as use the report viewer control. The custom viewer will use URL commands to integrate an SSRS 2012 report into a .NET Windows Forms–based application and the report viewer control will be used in an ASP.NET web page.

Integrating SSRS 2012 with .NET Applications

Now that you have some understanding of how URL access works, you'll learn how you can render reports. You'll start by looking at how to render a report by building a Windows Forms SSRS 2008 viewer application that uses the .NET 2.0 WebBrowser control and URL access to render reports for your application.

Building a custom Report Viewer Using the WebBrowser Control

We'll show how to create a simple Windows Forms application that contains the embedded browser, which you'll use to view a report.

Creating the Viewer Form

In this example, you will use C# to create the viewer application. We will also include the same application written in VB.NET in the extra downloadable content. Follow these steps:

1. Open Visual Studio 2010.
2. Create a new project.
3. On the New Project dialog box, select Visual C# ► Windows ► Windows Forms Application. Name it SSRS Viewer WBC. Under Solution, select Create New Solution, and name it Chapter 9. Click OK to create your project.

Now that you have created the project, you'll work with the form you will use to create the report viewer:

1. Visual Studio 2010 will add a single form to the project called Form1.cs. Rename the form to ViewerWBC.cs.
2. Resize the blank form to 800×600.
3. Add the WebBrowser control from the Toolbox's Common Controls section to the form. Name it WebBrowser, and anchor the control to the form's top, bottom, left, and right. Size the WebBrowser control to extend from both sides of the form to the bottom of the form.
4. Now add a textbox, and name it reportURL. Just after the textbox, add a button named reportRun, and set the button text to Run.

You should now have a form that looks like Figure 9-1.

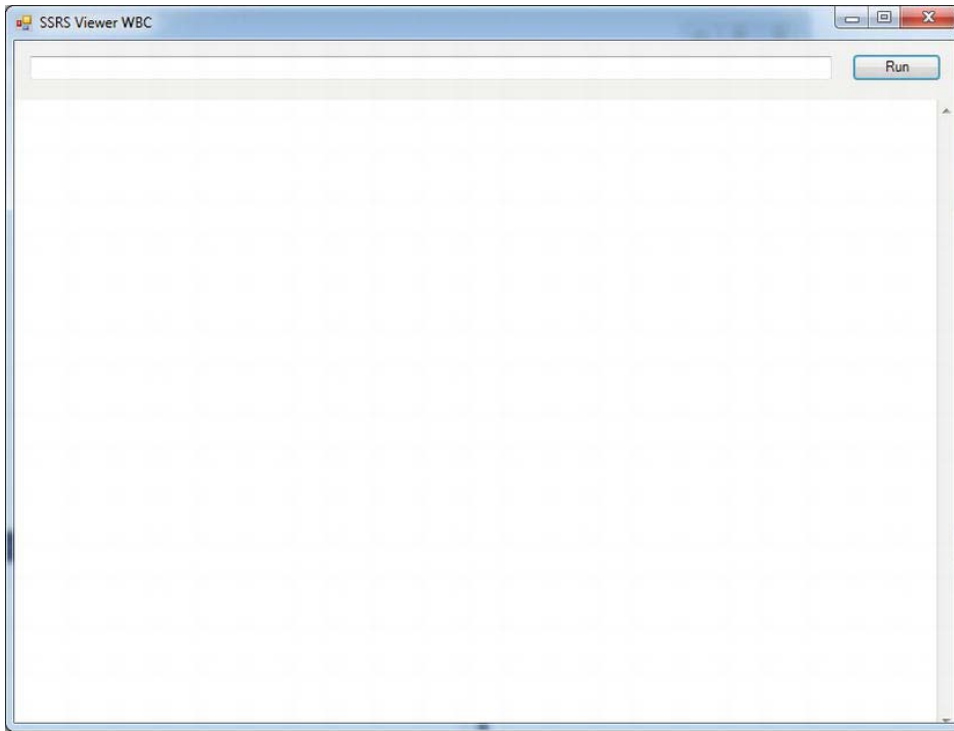


Figure 9-1. The *ViewerWBC.cs* form

Coding the Viewer Form

To code the viewer form, you'll add the code necessary to use this custom report viewer to render the SSRS 2012 reports. You need to add some code to the button's click event to make sure you can browse to and view an existing report. Make sure the *ViewerWBC.cs* form is open in design view, and double-click the *Run* button. This will create an empty method to handle the button's click event. Add the code in Listing 9-1 inside the *reportRun_Click* method to your method.

Listing 9-1. *reportRun* click Event: Browsing to URL

```
private void reportRun_Click(object sender, EventArgs e)
{
    webBrowser.Navigate(reportURL.Text);
}
```

Now run the project in debug mode. When the form displays, enter the following URL into the textbox:

```
http://localhost/reportserver/?/Pro_SSRS/Chapter_9/EmployeeServiceCost&rs:Command=Render
```

Now click Run. This renders the Employee Service Cost report. You will be prompted to supply parameters once the report begins to render in the web browser object. You may also need to use the name of your report server where you see localhost in the preceding URL. At this point, you should see something like Figure 9-2.

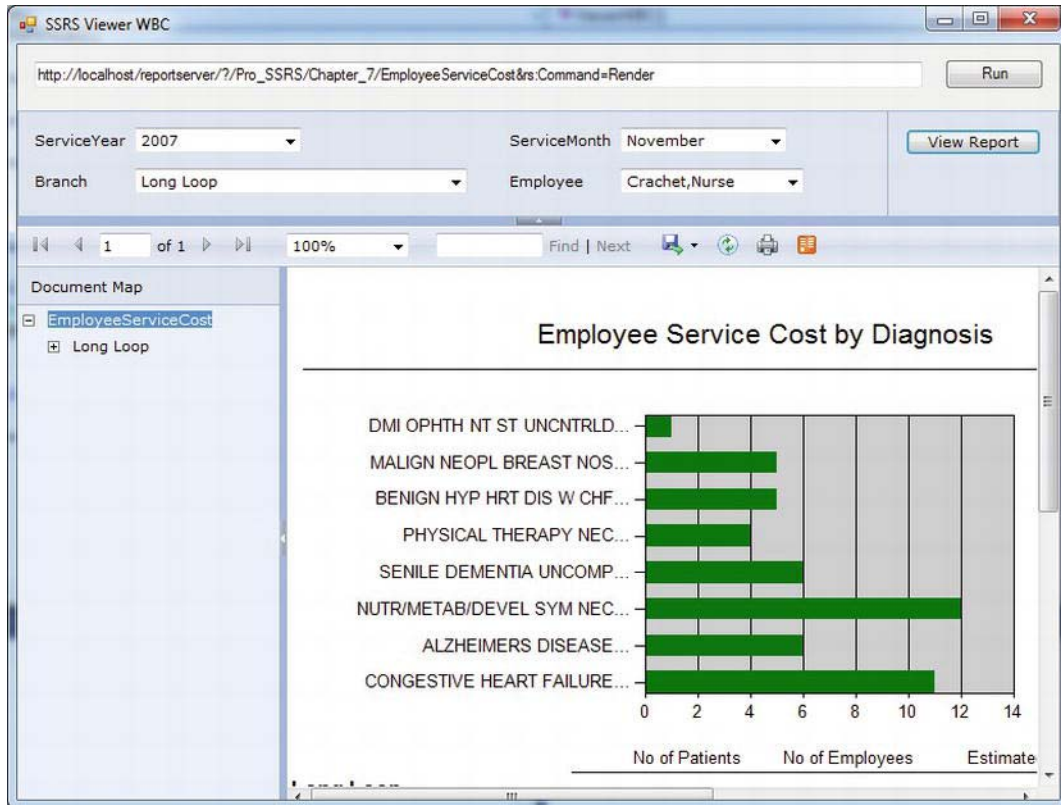


Figure 9-2. SSRS Viewer WBC application running the Employee Service Cost report

Next, we will add some code so that when the program is started it will set up a default string in the URL textbox with a path to the report, so you don't have to type it in each time. Right click on your ViewerWBC.cs file and select view code to pull up the section where you need to add this code.

```
public ViewerWBC()
{
    InitializeComponent();
    // Setting the initial URL for convenience in the example
    reportURL.Text = "http://localhost/reportserver/?rs:Command=Render";
    /Pro_SSRS/Chapter_9/EmployeeServiceCost&rs:Command=Render";
}
```

This is a very simple application and uses simple controls that every user should have available on their client. You can use the combination of this type of viewer with the URL commands we have discussed to include a custom report viewer in any application that you might build. Next, we will use a

less common – but more powerful – control to integrate our SSRS reports into a Windows form or ASP.NET project.

Building the Report Viewer Using a Report Viewer Control

Now, you'll learn how you can use the Report Viewer control to render the same Employee Service Cost report. Not only is it likely to be the most popular method for rendering server-based reports, but it also includes the ability to render reports based on the RDL locally without a report server.

You can use the two Report Viewer controls: one for Windows Forms applications and one for ASP.NET applications. These controls allow you to add rich reporting features to your Windows or ASP.NET applications more easily than is possible using any other method. The controls offer the power of the URL rendering method but make it easier to implement in your code because most options are now set as properties on the Report Viewer control. In other words, you don't need to include them in the string that you pass in as the URL. Another main advantage of working with the controls is that you get full IntelliSense support for all the options within the Visual Studio 2010 IDE.

Table 9-5 lists some of the key properties and methods of the Report Viewer controls.

Table 9-5. *Sample Report Viewer Properties and Methods*

Property/Method	Description
ProcessingMode	Determines whether the report is processed locally or by an SSRS 2012 server.
ServerReport.ReportServerUrl	Specifies the URL of the reporting server it will use to render a report when in remote mode.
ServerReport.ReportPath	Specifies the path to the specific report you want to render. Do not include the report server URL as part of the path since it is defined by ReportServerUrl.
LocalReport.ReportEmbeddedResource	Specifies the name of the report to use. By default when you add a local report (rdlc) to a project, Visual Studio 2010 sets the report as an embedded resource. Other properties allow you to point to a path on disk or even a stream for the report definition.
LocalReport.DataSources.Add	Specifies the method used to add a data source to the report.
RefreshReport	Causes the report to be rendered.

■ **Note** See the Visual Studio 2010 help for more information about the members of the Report Viewer control as well as for information about a number of other properties, methods, and events that come with the Report Viewer controls.

In this example, we will cover how to use the Windows Forms version of the Report Viewer control.

■ **Note** In the “Using the Report Server Web Service” section, we will show how to extend the application so that it uses Report Server Web service calls to query the report server for the parameters the report can accept and the values that are possible for each. We’ll show how to use these parameters to create dropdown lists for the users of the Windows Forms application and then render the report.

Creating the Viewer Form

To create the viewer form in C#, you’ll start by adding a new project to your solution by taking the following steps:

1. Select File ► Add ► New Project from the menu.
2. In the New Project dialog box, select Visual C# ► Windows ► Windows Forms Application. Name it SSRS Viewer RVC.
3. Now that you have created the project, you’ll work with the form that will create your report viewer:
4. Name the default form that was just created to ViewerRVC.cs.
5. Open the ViewerRVC.cs file in design mode by double clicking the file in Solution Explorer
6. Resize the blank form to around 800×600 pixels and change the text of the form to SSRS Viewer RVC in the properties section.
7. Add the Report Viewer control from the Toolbox’s Reporting section to the form. Name the control reportViewer in the properties, and anchor it to the form’s top, bottom, left, and right. Resize the control to fill the form but leave room at the top for a few more components. (If you do not have the report viewer control, you may need to download it from: www.microsoft.com/download/en/details.aspx?id=6610)
8. Now add a textbox, name it reportURL, and just after the textbox add three buttons named runServer, runLocal, and getParameters. Set their Text properties to Run Server, Run Local, and Parameters, respectively.

You should now have a form that looks like Figure 9-3.

■ **Note** In Visual Studio 2010 you will need to use a .NET framework of 3.5 or higher to get the ReportViewer controls in the IDE. If you use anything older than this, you will not get the option of adding a viewer to your project from the toolbox.

You'll now start adding the code necessary to use the new Report Viewer control to render the SSRS 2012 reports.

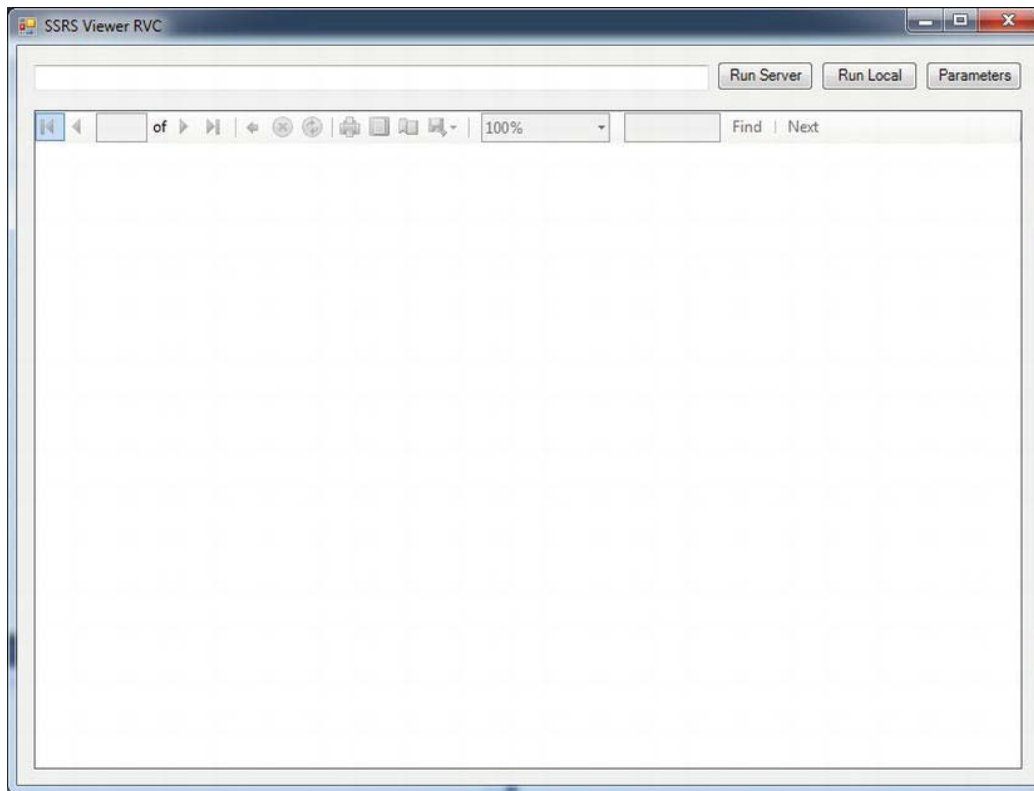


Figure 9-3. The ViewerRVC.cs form

Coding the Viewer Form

First, add a using statement for the new namespace, `Microsoft.Reporting.WinForms`, to the top of the class with the other namespace declarations. Doing this will allow you to access the members of the namespace without typing the full namespace each time you use a method or property from that namespace. Right click on the `ViewerRVC.cs` file and select view code to add this using statement to the file.

```
using Microsoft.Reporting.WinForms;
```

Second, add the code shown in Listing 9-2 to the Run Server button's click event. Make sure the `ViewerRVC.cs` form is open in design view, and double-click the Run Server button. This will create an empty method to handle the button's click event.

Listing 9-2. *The runServer Click Event: Running Report in Remote Mode*

```
private void runServer_Click(object sender, EventArgs e) {
    reportURL.Text = "/Pro_SSRS/Chapter_9/EmployeeServiceCost";
    reportViewer.ProcessingMode =
        sProcessingMode.Remote;
    reportViewer.ServerReport.ReportServerUrl = new Uri(@"http://localhost/reportserver/");
    reportViewer.ServerReport.ReportPath = reportURL.Text;
    reportViewer.RefreshReport(); }
```

Now run the project in debug mode. Then click Run Server. This renders the Employee Service Cost report where you can input parameters for your report. Of course, you need to use the name of your report server where you see localhost in Listing 9-2. At this point, you should see something that looks like Figure 9-4.

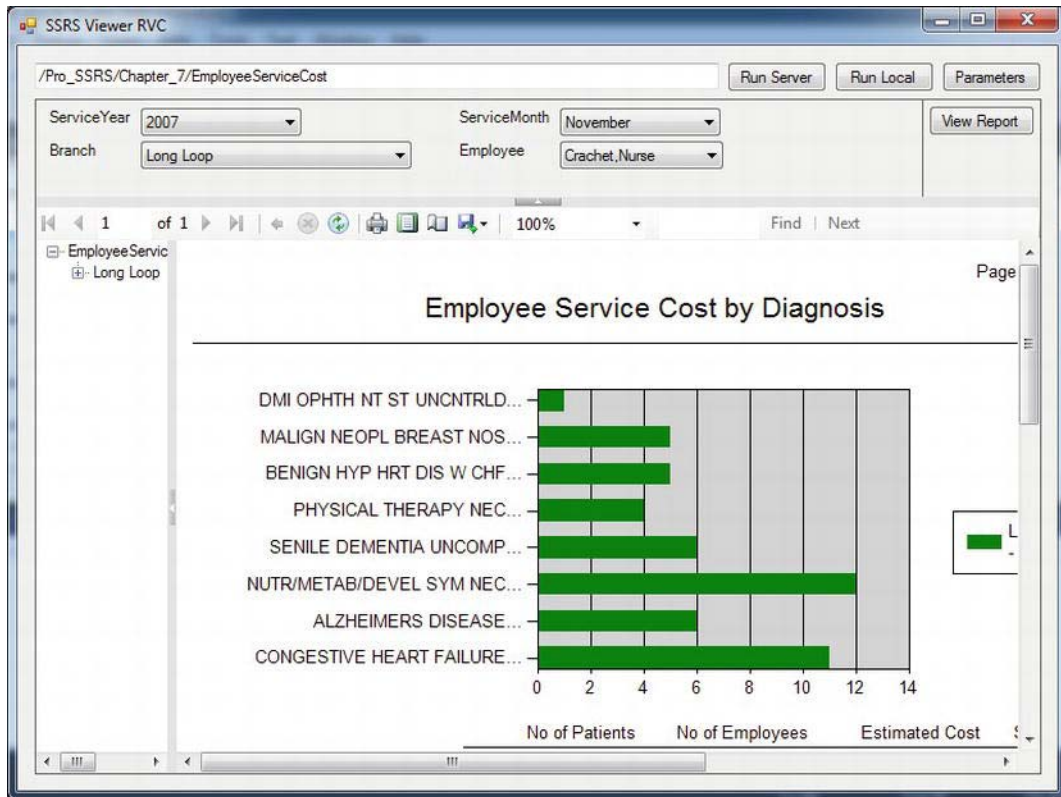


Figure 9-4. SSRS Viewer RVC running a server rendered report

That's all there is to rendering a report on an SSRS 2012 server using the Report Viewer control.

Rendering the Report Locally

We'll now cover how to render the report locally without using an SSRS 2012 server. It is a bit more complicated because you're responsible for filling the data sources with the data that your reports require. However, it does provide tremendous flexibility to the developer because you can use SSRS 2012 reporting features without a server.

■ **Caution** The server and local report definition files are not 100 percent compatible. This means it is not possible to use the same RDL source file for both local and remote use. In fact, you will note that by default reports for server-side or remote processing are created with the .rdl extension and reports for client-side or local processing are created with the .rdlc extension. While they are not 100 percent interchangeable, you can use one for the other with some changes to the underlying RDL.

Creating the Report's Data Source

First, you will add a data source to your project by adding the dataset you created previously. To add the data source, follow these steps:

1. In the Project Explorer, right-click the project name SSRS Viewer RVC, point to Add, and select Existing Item.
2. Navigate to the folder containing the Chapter 9 examples, and select EmployeePay.xsd in the SRSS Viewer RVC folder. This is the XML schema file you'll use to define your data source for the report. When you add the .xsd file to your project, it will also appear under the Data Sources window. If you don't see the Data Sources window, select Data ► Show Data Sources from the menu.

■ **Note** Alternatively, you could have used the Data Source Configuration Wizard found by selecting Data ► Add New Data Source. With it you can create several types of data sources including those sourcing from a database, a Web service, or even an object. Just remember that, much like this example, it is up to you to retrieve the data and populate the data source with it.

Next you'll create a local RDL (.rdlc) that you'll render using the Report Viewer control in local processing mode.

Designing the Report

To design the report, follow these steps:

1. To add a report to the project, right-click the project name—SSRS Viewer RVC—in the Solution Explorer.
2. On the shortcut menu, select Add ► New Item. This opens the Add New Item dialog box.
3. Select the Reporting section on the list to the left, then click the Report icon, enter **EmployeePay.rdlc** for the file name, and then click Add. This launches the Report Designer functionality in Visual Studio. The .rdlc extension signifies that it is a report for client or local rendering.
4. Make sure the report is selected. Open the Toolbox. From the Toolbox, drag a Table report item onto the report.
5. You should now be presented with the Dataset Properties window. From this window, select the Employees option in the Data Source drop down list. You will see the fields from this data source populated in the Fields data grid to the right. Name this dataset Employees_EmployeePay and click OK.
6. You should now see the table in your report designer. In the Data row of the first column, click the field icon in the top right hand corner to select a field. In the first column select the Employee ID field.
7. Perform this task again for the second and third columns. Choose StartDate and Amount for the second and third column, respectively.
8. Add a textbox to the top of the report and use the text “Employee Pay Report” for the value.

You should now have a report that looks like Figure 9-5.

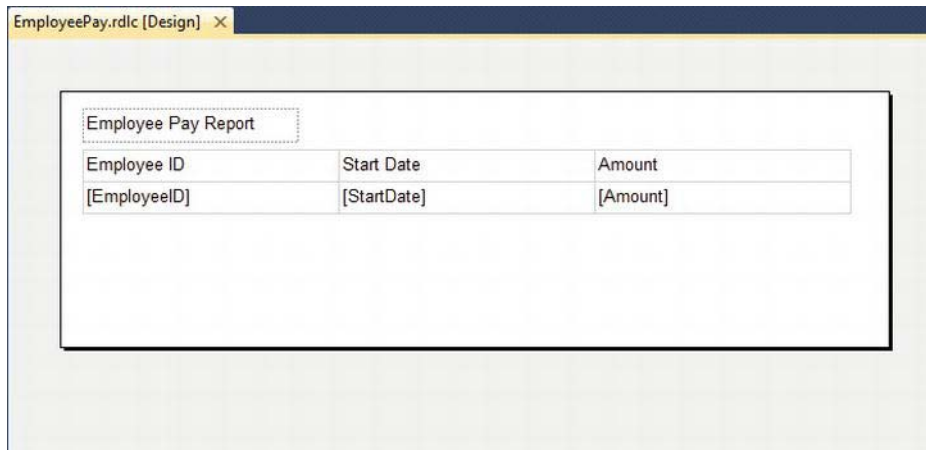


Figure 9-5. Local report in the designer

Now you have a report that uses the dataset you created as a data source. Add the code shown in Listing 9-3 to the `runLocal` button's click event to populate the dataset and display the values in the Report Viewer control using the `.rdlc` file you have created. Make sure the `ViewerRVC.cs` form is open in design view, and double-click the Run Local button. This will create an empty method to handle the button's click event.

Listing 9-3. *runLocal click Event: Running Report in Local Mode*

```
private void runLocal_Click(object sender, EventArgs e)
{
    Employees empDS = new Employees();
    empDS.ReadXml(@"C:\Temp\EmployeePay.xml");
    reportViewer.ProcessingMode = ProcessingMode.Local;
    reportViewer.LocalReport.ReportEmbeddedResource = "SSRS_Viewer_RVC.EmployeePay.rdlc";
    reportViewer.LocalReport.DataSources.Add(new ReportDataSource("Employees_EmployeePay",
    empDS.Tables["EmployeePay"]));
    reportViewer.RefreshReport();
}
```

■ **Note** The `ReportEmbeddedResource` value must include the namespace of your project. If you entered spaces for the name of your project when you created it, the namespace will use underscores in place of each space.

Now run the project, not the whole solution, in debug mode. Remember you can do this by right clicking the specific project and selecting `Debug -> Start new instance`. When the form displays, click `Run Local`. This renders the `localEmployee Pay` report. At this point, you should see something like Figure 9-6. You've now created a report viewer in a Windows Forms application by using the `WebBrowser` control with URL rendering and by using the new `Report Viewer` control. You could stop at

this point and just use these methods to render the reports and to display the report parameters, toolbar, and report. However, in this example, you want to use the SSRS 2012 Report Server Web service to get a list of parameters for the selected report and to display them to the user in a Windows Form. To get this list of parameters, you need to call the `GetReportParameters` method on the SSRS 2012 Report Server Web service.

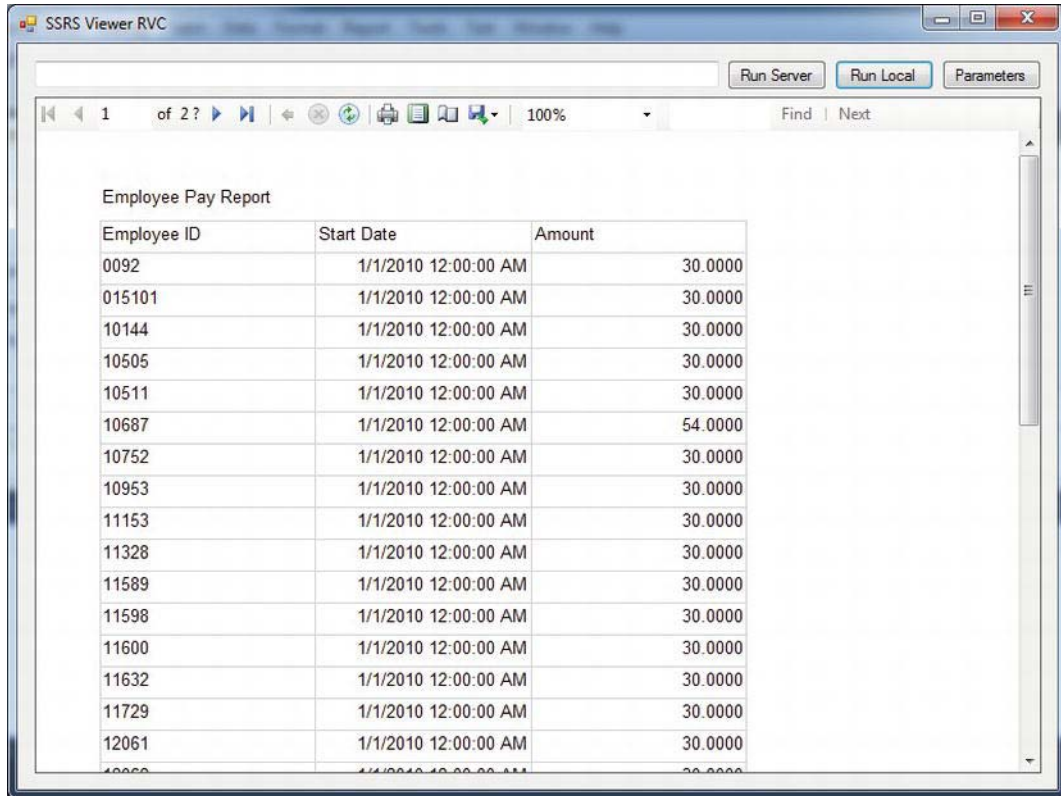


Figure 9-6. SSRS Viewer RVC running a locally rendered report

Using the Report Server Web Service

The SSRS 2012 Web service uses the SOAP API to allow you to call a variety of methods on the report server and interact with them using a rich set of objects provided by the service.

There are two services that you should be concerned with in SSRS 2012. The first, `ReportingService2010`, handles the management of the report server and the reports within. We will be using this service for the remainder of the chapter. The second service, `ReportExecutionService`, handles the programmatical rendering of reports without the use of a browser or report viewer.

Web Services Method Categories

The Report Server Web services can control every aspect of the report server and consists of several categories of methods, as listed in Table 9-6.

The Report Server Web services uses many of these methods to control aspects of SSRS 2012 that aren't directly related to controlling reports, so we won't cover them in this chapter. However, you should be aware of the level of control your custom application can have over SSRS 2012 and the types of functions that can be performed, because you might want to provide a user interface to them from your application. Keep in mind that Microsoft built the main SSRS 2012 report manager application using ASP.NET and these Web services.

For the SSRS Viewer RVC, you are using the Report Viewer control to render the report, but you want to provide a custom Windows Forms–based user interface to allow users to enter their report parameters. The rest of this chapter concentrates on using methods from the report parameters category listed in Table 9-6. You'll use these methods to obtain a list of the parameters that the report expects and also to find the possible values for those parameters.

Table 9-6. Categories of the Report Server Web Services Methods

Category	Manages
Namespace management	Folders and items on the server and their properties
Authorization	Tasks, roles, and policies
Data source connections	Data source connections and credentials
Report parameters	Setting the retrieval parameters for reports
Models	Managing model creation, removal, modification and policies
Rendering and execution	Report execution, rendering, and caching
Report history	Snapshot creation and history
Scheduling Shared	schedule creation and modification
Subscription and delivery	Subscription creation and modification
Linked reports	Linked report creation and management

You'll use this information to create and populate combo boxes that allow the users to enter their choices from a Windows Forms dialog box.

Creating the GetParameters Form

You already have a form (ViewerRVC.cs) to render and display the report in the embedded Report Viewer control. You'll now add a second form to the project that you'll use to display the report and rendering parameters and to allow the users to make their selections.

So, add a `GetParameters` form to your project by selecting **Project ► Add Windows Form**. Name the form `GetParameters.cs` and modify the title text of the form to `Parameters`. Onto this form, add two controls from the Toolbox; add a `FlowLayoutPanel` control named `parameterPanel` from the Containers section, and add one button control named `buttonOK` in the button properties window. Then set the text property for the button to `OK` in the button properties window. When you're done, the form should look like Figure 9-7.

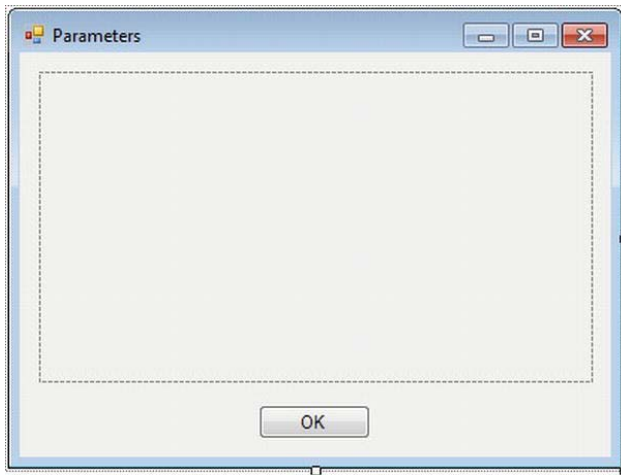


Figure 9-7. SSRS Viewer RVC `GetParameters` form

You now need to add a reference to the SSRS 2012 Web service to your project. You do this by selecting **Project ► Add Service Reference** or by right clicking the Service References in the Solution Explorer and selecting **Add Service Reference**. When the dialog box appears click the **Advanced** button at the bottom of the form. On this next form, click the **Add Web Reference** button. From here, enter the following URL:

```
http://localhost/reportserver/reportservice2010.asmx
```

Substitute the name of your server for `localhost`, if need be. Then click the **Go** button, which is the right arrow icon. You'll see a dialog box similar to Figure 9-8.

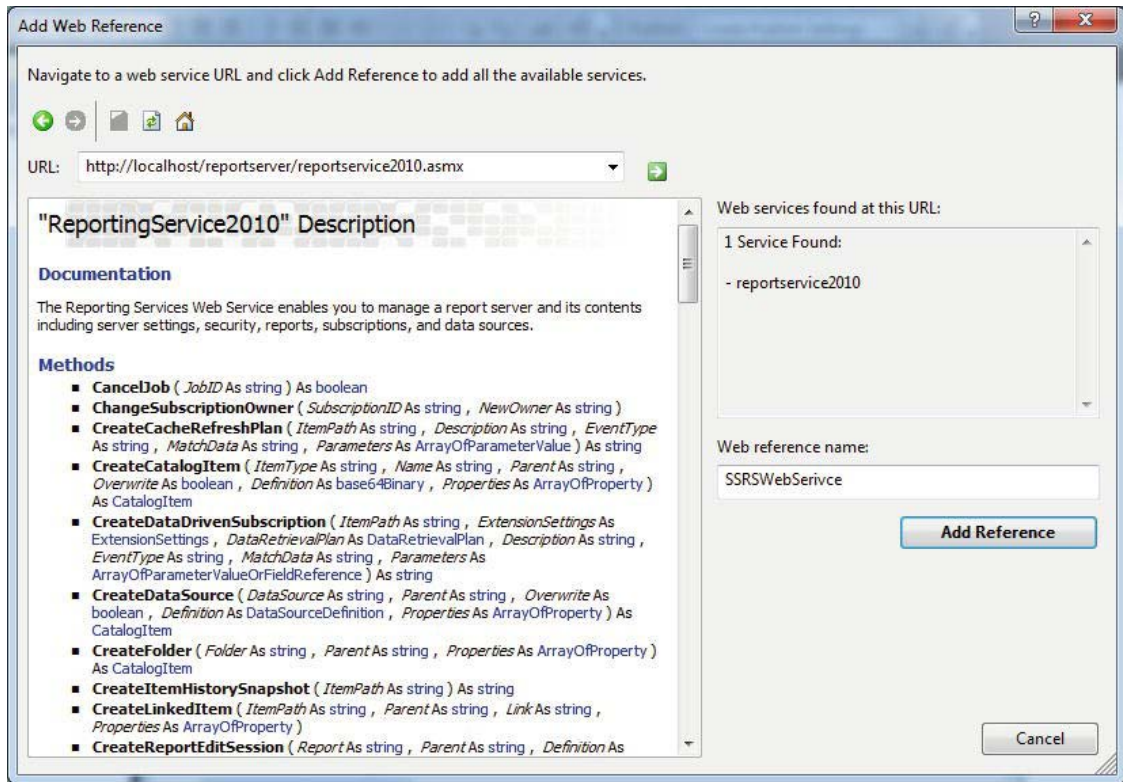


Figure 9-8. Adding an SSRS 2008 Web service reference

In the Web Reference Name textbox, enter **SSRSWebService**, which is the name by which you'll refer to your Web service in your code. From here, click the Add Reference button to finalize the reference addition.

Once this dialog box is closed, you will add the following using statement to the top of your code in the GetParameters class file. Note that if your project name is different from SSRS Viewer RVC, you should change the reference to reflect the name of your project.

```
using SSRS_Viewer_RVC.SSRSWebService;
```

Now you can reference the methods and properties exposed by the Web service much more easily because you won't have to enter the fully qualified namespace. Also, add the following other directives underneath the one you just added:

```
using System.Web.Services.Protocols;
using System.Collections;
```

Doing this will allow you to access the members of these namespaces without needing to type the full namespace each time you use a method or property from that namespace.

Because you'll use this form as a dialog box to display the report parameters and their possible values, and allow the user to select them, you need a way to communicate between the main form and this newly created GetParameters form. We will set that communication up in the next section.

Coding the Report Parameters Form

When you instantiate the report parameters form (GetParameters.cs), you do so by passing in the URL of the report the user has entered in the viewer form (ViewerRVC.cs). You use this URL to determine the report server name and the specific report the user wants to run. You need to know both of these for the calls to the Report Server Web service. Because you'll use this information throughout the rest of the code, in your GetParameter class you will store them in some class-level private variables, as shown in Listing 9-4. Add the list of private variables to your GetParameter.cs class code directly after the class start.

Listing 9-4. *Class-Level Private Variables*

```
public partial class GetParameters : Form
{
    private string url;
    private string server;
    private string report;
    private Microsoft.Reporting.WinForms.ReportParameter[] parameters;
    ReportingService2010 rs;
```

In the form constructor, which will take the URL string as the only parameter, we will break the report server and report name into two separate fields. To break down the URL into the server and report name, use the string split method and create a constructor that looks like Listing 9-5. Add this constructor code, the code that is first run when an object of a class type is created, directly below the private variables. You will need to add the string parameter called URL to the constructor as well.

Listing 9-5. *Report Parameters Form Constructor*

```
public GetParameters (string URL)
{
    InitializeComponent();
    url = URL;
    string[] reportInfo = url.Split('?');
    server = reportInfo[0];
    report = reportInfo[1]; }
```

The GetParameters_Load Event

Now you get to where the real work for this dialog box takes place: the Form_Load event. To create this method automatically, open your GetParameters form in display mode and double click your form where there are no components, such as the top bar. This will open your form in code view and already have the GetParameters_Load method created and link it to the form's load event.

Next, create a ReportingService object so that you can access SSRS 2012 through the Web service that you added as a reference earlier. Then, set your Windows credentials as the credentials to be used for calling the Web service, like so:

```
rs = new ReportingService2005();
rs.Credentials = System.Net.CredentialCache.DefaultCredentials;
```

■ **Note** You can also use Basic authentication using `rs.Credentials = new System.Net. NetworkCredential("username", "password", "domain");`. The method you use depends on the security settings for the report server virtual directory. By default, it's configured to use Windows authentication.

Calling the Web Services GetItemParameters Method

The `GetItemParameters` method takes five parameters:

- **Item:** The full path name of the report/item.
- **ForRendering:** A Boolean value that indicates how the parameter values should be used. You must set it to true to get a list of the possible values for each parameter.
- **HistoryID:** The ID of the report history snapshot. You set it to null because you aren't running the report from a snapshot.
- **ParameterValues:** The parameter values (`ParameterValue[]` objects) that can be validated against the parameters of the report that are managed by the report server. Set this to null for this example.
- **Credentials:** The data source credentials (`DataSourceCredential[]` objects) that can be used to validate query parameters.

The `GetItemParameters` method returns an array of `ItemParameter[]` objects that contain the parameters for the report. You use this information to render combo boxes that allow the users to select the parameter values they want to use when running the report.

The following code sets up the variables you need to use and then calls the `GetItemParameters` method to retrieve a list of parameters that the report expects, as shown in Listing 9-6.

Listing 9-6. Call to GetReportParameters

```
bool forRendering = true;
string historyID = null;
ParameterValue[] values = new ParameterValue[1];
DataSourceCredentials[] credentials = null;
ItemParameter[] parametersSSRS = null;
parametersSSRS = rs.GetItemParameters(report, historyID,
    forRendering, values, credentials);
```

Once you have the list of parameters from SSRS 2012, you loop through them using the values to create labels as you create your combo box for each parameter, like so:

```
foreach (ItemParameter rp in parametersSSRS)
```

Each `ItemParameter` object has a read-only property called `ValidValues`. You can use the `ValidValues` property, which returns an array of `ValidValue` objects to populate the items in each combo box, as shown in Listing 9-7.

Listing 9-7. Iterating Through the ValidValue Objects

```
if (rp.ValidValues != null) {
    //Build list items
```

```

ArrayList alist = new ArrayList();
pvs = rp.ValidValues;
foreach (ValidValue pv in pvs)
{
    alist.Add(new ComboItem(pv.Label,pv.Value));
}
//Bind list items to combo box
a.DataSource = alist;
a.DisplayMember="Display";
a.ValueMember="Value"; }

```

So, for each ReportParameter, you see whether any ValidValues properties exist. If they do exist, you loop through them, adding each item to the combo box. Because you want to retrieve the display name and the actual value for each item in the combo box, you have to create a combo box item class and bind the objects to the combo box. Listing 9-8 shows the full GetParameters_Load event method, and Listing 9-9 shows the ComboItem class.

Knowing something about our report, we already know that some of the parameters are dependent on other parameter values. If we simply run through the list of ValidValues for each parameter, some of them will include no values because of this dependency. We can use this previous knowledge to pass in a bit of information to the getItemParameters method. We will place one ParameterValue item in the values array so we can retrieve some valid values for all of our other parameters.

We also need to retrieve the display name and the value selected of each item in the combo boxes. We are also going to create a combo box item class to handle this by binding each combo item to the combo box. Listing 9-8 shows the GetParameters_Load event and Listing 9-9 shows the ComboItem class.

■ **Note** We are populating the rest of the parameters based on just one possible Service Year value. Since, in some reports, the other parameter valid values might change, you could include an overridden event method that would refresh each of the other parameter values anytime the year was changed.

Listing 9-8. *Get Report Parameters and Possible Values and Display Them in Combo Boxes*

```

private void GetParameters_Load(object sender, EventArgs e)
{
    rs = new ReportingService2010();
    rs.Credentials = System.Net.CredentialCache.DefaultCredentials;

    bool forRendering = true;
    string historyID = null;
    ParameterValue[] values = new ParameterValue[1];
    DataSourceCredentials[] credentials = null;
    ItemParameter[] parametersSSRS = null;
    ValidValue[] pvs = null;

    int x = 5;
    int y = 30;

```

```

        try
        {
            values[0] = new ParameterValue();
            values[0].Label = "ServiceYear";
            values[0].Name = "ServiceYear";
            values[0].Value = "2009";

            parametersSSRS = rs.GetItemParameters(report, historyID,
forRendering, values, credentials);

            if (parametersSSRS != null)
            {
                foreach (ItemParameter rp in parametersSSRS)
                {
                    this.SuspendLayout();

                    this.parameterPanel.SuspendLayout();
                    this.parameterPanel.SendToBack();

                    // now create a label for the combo box below
                    Label lbl = new Label();
                    lbl.Anchor = (System.Windows.Forms
.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Left);
                    lbl.Location = new System.Drawing.Point(x, y);
                    lbl.Name = rp.Name;
                    lbl.Text = rp.Name;
                    lbl.Size = new System.Drawing.Size(150, 20);
                    this.parameterPanel.Controls.Add(lbl);
                    x = x + 150;

                    // now make a combo box and fill it
                    ComboBox a = new ComboBox();
                    a.Anchor = (System.Windows.Forms
.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Right);
                    a.Location = new System.Drawing.Point(x, y);
                    a.Name = rp.Name;
                    a.Size = new System.Drawing.Size(200, 20);
                    x = 5;
                    y = y + 30;

                    this.parameterPanel.Controls.Add(a);
                    this.parameterPanel.ResumeLayout(false);
                    this.ResumeLayout(false);

                    if (rp.ValidValues != null)
                    {
                        //Build listitems
                        ArrayList alist = new ArrayList();
                        pvs = rp.ValidValues;
                        foreach (ValidValue pv in pvs)
                        {
                            alist.Add(new ComboItem

```

```
(pv.Label, pv.Value));  
  
        }  
        //Bind listitmes to combobox  
        a.DataSource = alist;  
        a.DisplayMember = "Display";  
        a.ValueMember = "Value";  
    }  
}  
  
catch (SoapException ex)  
{  
    MessageBox.Show(ex.Detail.InnerXml.ToString());  
}
```

Listing 9-9. Combo Item Class

```
public class ComboItem
{
    public ComboItem(string disp, string myvalue)
    {
        if (disp != null)
            display = disp;
        else
            display = "";
        if (myvalue != null)
            val = myvalue;
        else
            val = "";
    }

    private string val;
    public string Value
    {
        get { return val; }
        set { val = value; }
    }

    private string display;
    public string Display
    {
        get { return display; }
        set { display = value; }
    }

    public override string ToString()
    {
        return display;
    }
}
```

Upon loading, you'll see a form like Figure 9-9, which displays a series of combo boxes, each containing the valid values for the report parameters. You won't be able to see this in full view until we add the code to open this parameter form from the main form. We will be performing that step shortly in the chapter.

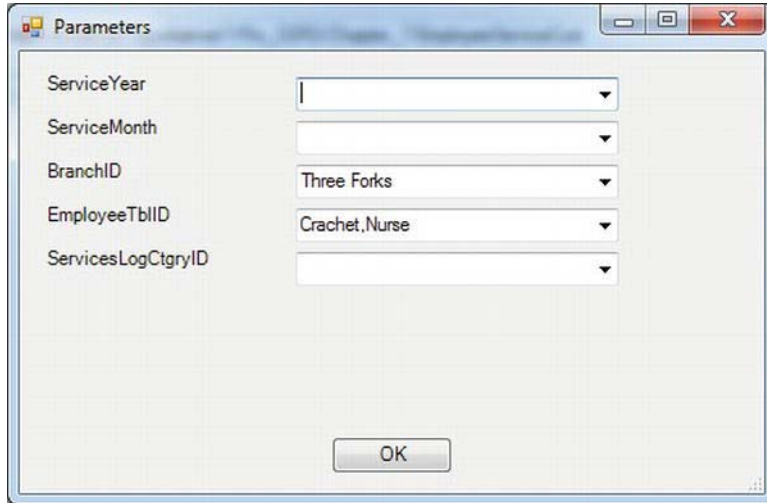


Figure 9-9. The Parameters dialog box

Rendering the Final Report

To finish your SSRS 2012 Windows Forms viewer application, you need to set the parameters' local variables to the parameter values the user has selected so that you can retrieve them from the ViewerRVC.cs form, using a property you will create in the GetParameters class called Parameters. You will populate the parameters variable by creating a method called ViewerParameters.

Make sure the GetParameters form is displayed in design mode, and double-click the OK button. This will create a new event handler method called buttonOK_Click automatically and set it up to be run anytime the OK button on our form is pressed. To the buttonOK click event handler, add the code shown in Listing 9-10.

Listing 9-10. The buttonOK click Event Handler for the OK Button

```
private void buttonOK_Click(object sender, EventArgs e) {
    parameters = ViewerParameters();
    this.DialogResult = DialogResult.OK;
    Close(); }
```

The method that it calls, ViewerParameters, simply iterates through the combo boxes and creates an array of ReportParameters, which are essentially name-value pairs used by the Report Viewer control, as shown in Listing 9-11. You will need to add this entire method to your GetParameters class manually. Let's discuss what this method does before as well.

Our ViewerParameter method first counts the number of controls that hold values and then initializes an array of report parameters to be used as the return value. It then looks through each control inside of the parameter panel we set up and tests if that control is a combo box, which is the type we

want to get the values out of to return. If the control is a combo box, the code then casts that control as a combo box so that we can grab the name and value of the selected item to add to our array of report parameters. If the value isn't null or a blank string, we grab the selected value and populate the array. If it is null or blank, we use a null value in our parameter array.

Listing 9-11. *Get Parameters Entered by User*

```
private Microsoft.Reporting.WinForms.ReportParameter[] ViewerParameters() {
    int numCtrls = (this.parameterPanel.Controls.Count / 2);
    Microsoft.Reporting.WinForms.ReportParameter[] rp =
    New Microsoft.Reporting.WinForms.ReportParameter[numCtrls];
    int i = 0;
    foreach (Control ctrl in this.parameterPanel.Controls) {
        if (ctrl.GetType() == typeof(ComboBox)) {
            ComboBox a = (ComboBox)ctrl; rp[i] =
            new Microsoft.Reporting.WinForms.ReportParameter(); rp[i].Name = a.Name; if (a.SelectedValue
            != null &&
            a.SelectedValue.ToString() != String.Empty) {
                rp[i].Values.Add(a.SelectedValue.ToString()); }
            else {
                rp[i].Values.Add(null); }
            i++; } } return rp; }
```

To finish your GetParameters form, you need to add some code to allow you to pass the report parameters and their values to the viewer form (ViewerRVC.cs), as shown in Listing 9-12. This allows the parent ViewRVC form to gather the parameters once they have been populated from the web service and chosen by you through the form. You will also manually add this code to your GetParameters class.

Listing 9-12. *Property Used to Get Parameters from the ViewerRVC.cs Form*

```
public Microsoft.Reporting.WinForms.ReportParameter[] Parameters {
    get
    {
        return parameters;
    } }
```

To finally bring it all together, you need to add a button and related code to its click event in the ViewerRVC.cs form, the main form, to use the new Parameters dialog box, passing in the URL of the report you want to run. Then, you read the ReportParameter array through the GetParameters form's Parameters property and use it to set the report parameters for the Report Viewer control, as shown in Listing 9-13.

First, open your ViewerRVC class in design mode by simply double clicking the file in the Solution Explorer. From here, you can just double click the Parameters button that you created when we first designed this form earlier in the chapter. This will create a new getParameters_Click event method for you and link it automatically to the click event for that button. Now you can add the code to pull up the new GetParameters form and return the parameters generated from that form.

Listing 9-13. *The getParameters click Event: Retrieving the Parameters and Running the Report*

```
private void getParameters_Click(object sender, EventArgs e)
{
    reportURL.Text = "http://localhost/reportserver? "
```

```

/Pro_SSRS/Chapter_9/EmployeeServiceCost";
GetParameters reportParameters = new GetParameters(reportURL.Text);
if (reportParameters.ShowDialog() == DialogResult.OK) {
    reportViewer.ProcessingMode =
    Microsoft.Reporting.WinForms.ProcessingMode.Remote;
    reportViewer.ServerReport.ReportServerUrl =
    new Uri(@"http://localhost/reportserver/");
    reportViewer.ServerReport.ReportPath =
    "/Pro_SSRS/Chapter_9/EmployeeServiceCost";
    reportViewer.ServerReport.SetParameters(reportParameters.Parameters);
    reportViewer.ShowParameterPrompts = false; reportViewer.RefreshReport(); } }

```

Now run the project in debug mode. When the form displays, click the button labeled Parameters. The Parameters dialog form will be displayed. Select some parameters from the available drop down values, and click OK. This renders the Employee Service Cost report, which is located on the SSRS 2012 server, using the parameters you have supplied and it won't prompt you to supply any additional parameters. At this point, you should see something that looks like Figure 9-10.

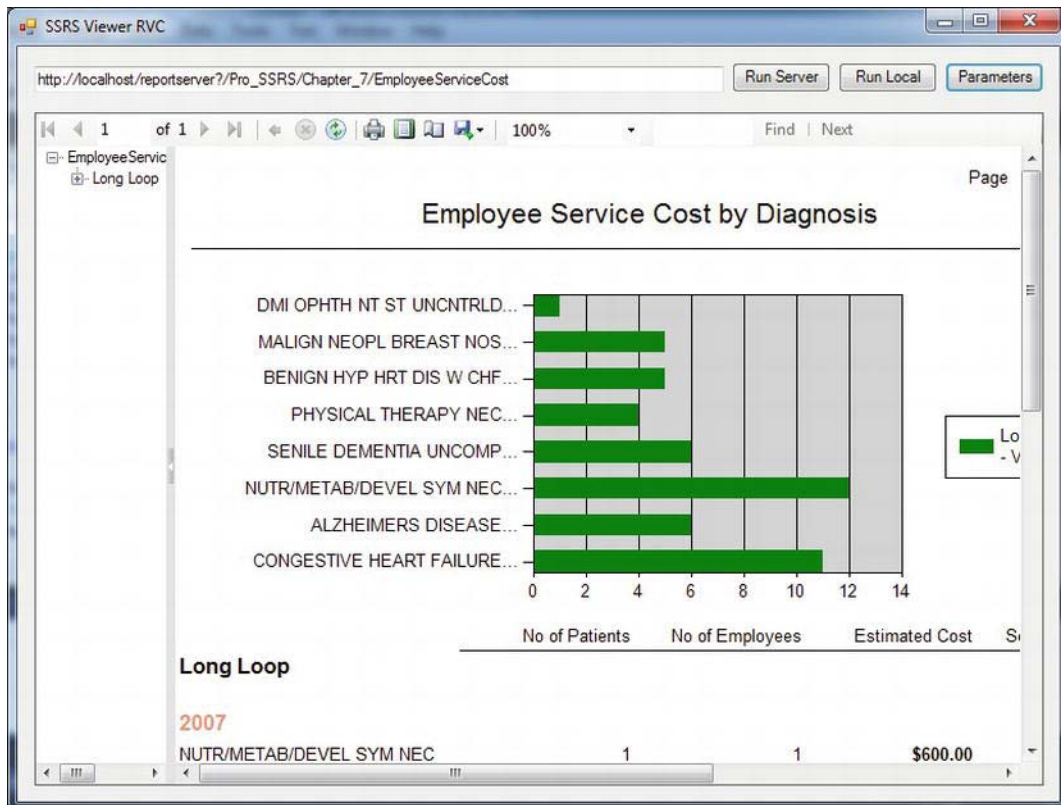


Figure 9-10. The finished report using your parameters

Now you have a foundation for using the report viewer control with Windows Forms applications. This chapter's examples have implemented URL access in a web browser control as well as the Report Viewer control for rendering the reports, providing you with multiple choices for integrating reporting into your Windows Forms applications. You've made your viewer a lot more user friendly by using SSRS 2012's SOAP-based API to access the rich functionality of SSRS to retrieve the available report parameters and possible values. This allows you to create a more familiar and responsive Windows-based user interface for your users.

You can also render reports using the Report Server Web service directly. However, you lose the functionality of features such as the report toolbar with its built-in navigation and export functionality. This means you have to create these on your own if you use the other web service for rendering.

Building the Report Viewer in ASP.NET

With Visual Studio, you are also given the option to use the Report Viewer control in an ASP.NET application. This can give you the flexibility to develop custom reporting applications that do not have to be distributed to users or updated on an end user's machine each time a patch is deployed. Users can access your custom application through any standard Web browser, and any modifications can be transparent to your end users.

You will use many of the same concepts and methods in this project that were used in the forms application that we have previously built. Start by creating a new ASP.NET Web Application project. Name this project SSRS_WebViewer and choose a location for your source files.

■ **Tip** Don't worry if you do not have IIS available on your development machine. Using the debugging features of Visual Studio will allow you to see your ASP.NET application in a Web browser.

You will now have a new Web application project with a Default.aspx file created for you. Open this Web page and add five controls to the layout of the page:

- *DropDownList*: You will use this drop-down list to hold the names of the reports returned from the SSRS 2012 Web service. Name it reportList.
- *Button*: This will be used to trigger the rendering of a report in the Report Viewer. Name it renderReport and change the display text to Render.
- *Horizontal Rule*: You will use a horizontal rule in the form just to separate the top controls from the Report Viewer.
- *Report Viewer*: You will render the reports stored on the SSRS 2012 server with this control. Name it reportViewerWeb.
- *ScriptManager*: It is a requirement of using the ReportViewer control that you include a ScriptManager control on your page. It makes sure that all the required prerequisite code is included in the page that is needed by the ReportViewer.

Once you have the controls on the page and sized to fit your layout, add an on-click action to the button that was just created. You can do this by viewing the page in design mode and double-clicking the button, just like a Windows form button. Don't worry about the contents of the function that gets

created; you will populate the code later. Once you are finished, your code should look similar to Listing 9-14.

Listing 9-14. *Default.aspx Partial Code Listing*

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

            <asp:DropDownList ID="reportList" runat="server">
            </asp:DropDownList>
            <asp:Button ID="renderReport" runat="server" onclick="renderReport_Click"
                Text="Render" />
            <hr />
            <rsweb:ReportViewer ID="reportViewerWeb" runat="server" Width="841px">
            </rsweb:ReportViewer>

        </div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
    </form>
</body>
</html>
```

Now that you have the page design finalized, you will add a reference to the SSRS 2010 management Web service. Right-click the parent folder for this project in the Solution Explorer and choose Add Web Reference. From this screen, input the address of the Web service we are going to use. In this case, you will be adding `http://localhost/reportserver/reportservice2010.asmx`. You may need to change the server name in this URL if you are not developing directly on your SSRS server. Name this Web service reference `SSRS_WebService`. You can see what this should look like in Figure 9-11.

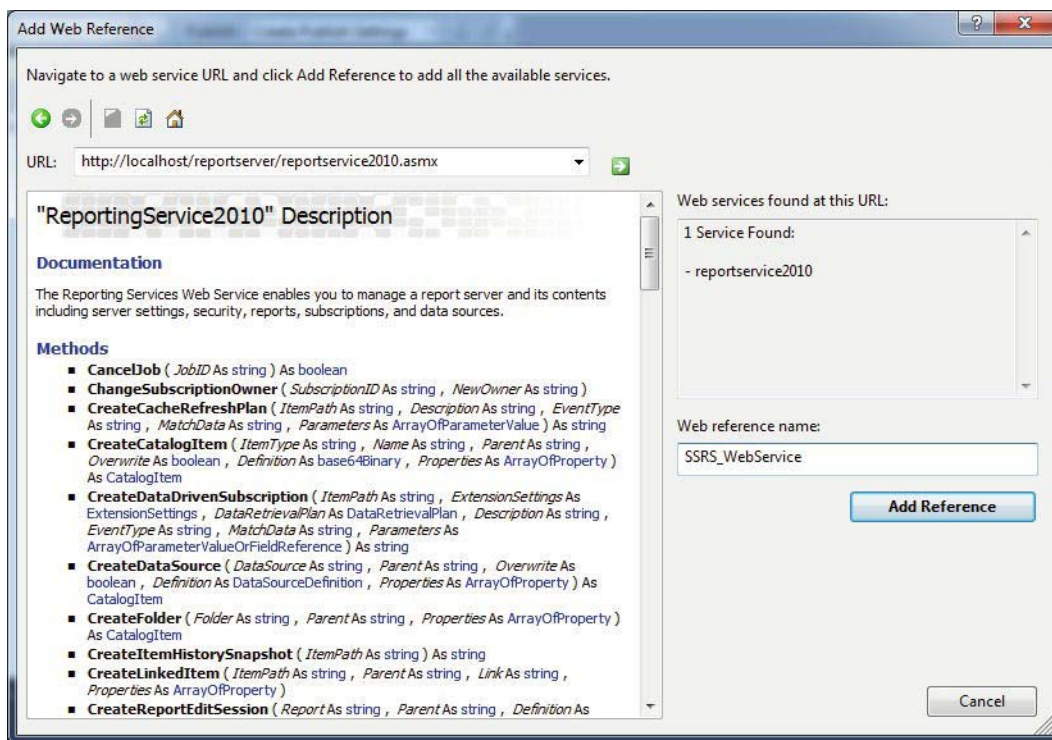


Figure 9-11. Adding a Web service reference

Now that you have the Web service ready to reference, you can populate the drop-down list with report items from the Chapter_9 directory of your SSRS instance. You should already have at least one report in that folder from the setup of this chapter.

You will use the Web service you have just registered to retrieve all of the report items from the Chapter_9 directory. The Web service method you will use to do this is ListChildren. This function takes two parameters: the path from where to gather the list from and a Boolean value to determine whether the search should be recursive. Since you want to get the reports from the Chapter_9 folder and nothing else, you want to pass only that path and a false value to get only the reports in this one directory.

You want to populate the drop-down list with only report items. However, ListChildren will return all the items in this folder, not just the reports. You will use the TypeName property to determine the type of the returned object and only list reports. Once you have tested whether the item is a report or not, you will populate the drop-down list with the names of the reports. The code to perform all of this exists in the Page_load method of the page, so that you start with a full drop-down list as soon as the page is done loading. Listing 9-15 shows the code used to populate the drop-down list from the Web service.

Listing 9-15. Using the SSRS Web Service to Populate the Report List

```
SSRS_WebService.ReportingService2010 rs = null;

protected void Page_Load(object sender, EventArgs e)
```

```

    {
        try
        {
            rs = new SSRS_WebService.ReportingService2010();
            rs.Credentials = System.Net.CredentialCache.DefaultCredentials;

            SSRS_WebService.CatalogItem[] listItems = rs.ListChildren(
"/Pro_SSRS/Chapter_9", false);
            foreach (SSRS_WebService.CatalogItem thisItem in listItems)
            {
                if (thisItem.TypeName == "Report")
                {
                    reportList.Items.Add(thisItem.Name);
                }
            }
        }
        catch (Exception ex)
        {
            Response.Write(ex.Message);
        }
    }
}

```

Now that you have the form built and the drop down list populated with data from the SSRS Web service, you can finalize the application by allowing the user to render the reports from the list. You will now go back to the generated code for the Click event of the button you added to the Web page. In this section, you will set the processing mode, the report server URL, and the path to the report you want to render. Finally, the code will render the report and it will be displayed in the Report Viewer control. Listing 9-16 shows the populated on click event in the code view of the web page.

Listing 9-16. *Populating the RenderReport Button Click Method*

```

protected void renderReport_Click(object sender, EventArgs e)
{
    reportViewerWeb.ProcessingMode = Microsoft.Reporting.WebForms.ProcessingMode
.Remote;
    reportViewerWeb.ServerReport.ReportServerUrl = new Uri(
(@"http://localhost/reportserver/");
    reportViewerWeb.ServerReport.ReportPath = "/Pro_SSRS/Chapter_9/" +
reportList.SelectedItem.Value;
    reportViewerWeb.ServerReport.Refresh();
}

```

To test the application, right click the project in the solution explorer and select Debug -> Start New Instance. Visual Studio will start a stand-alone Web server on a random port and launch the project for debugging. Choose the report that you put in the directory and click the Render button. If everything is set up correctly, after you select some parameters on the report, you should get an Internet Explorer window that resembles Figure 9-12.

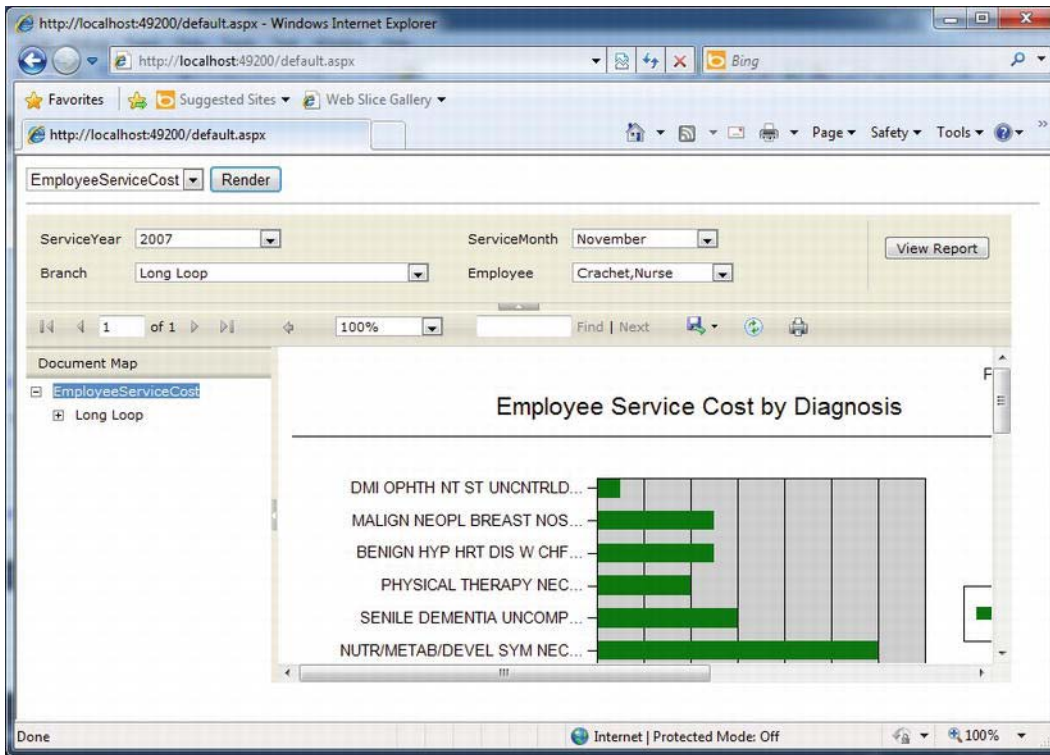


Figure 9-12. Finalized Web report application

Summary

In this chapter, we showed how to use the URL access capabilities of SSRS 2012 and the Report Viewer controls to embed reports quickly into applications. Beyond the WebBrowser and Report Viewer controls used in this chapter, you can use other applications to render reports. For example, you can integrate SSRS 2012 with Microsoft SharePoint Server. We'll discuss using this integration in greater detail in Chapter 12. By combining SharePoint and SSRS, you can quickly build a portal that displays your reports without much code at all.

In this chapter, you also learned how to make Report Server Web service calls to augment your Windows Forms viewer application. The application you created allows you to type the URL of a server-based report you want to view. It then uses SSRS 2012's `GetItemParameters` method to retrieve a list of report parameters and the `ValidValues` property to retrieve possible values. It then reads the values selected by the user and populates an array of `ReportParameters` which is then used by the Report Viewer control to render your report with the selected parameters already in place. In Chapter 10, we'll expand on this example by using the Report Server Web service to allow the user to set the report to run on a schedule with the supplied parameters instead of rendering it immediately.

Managing Reports

In many reporting solutions prior to SSRS, report management required little more than delivering the completed report file to the end user via a file share or embedded in a third-party application. SSRS is a full reporting environment with features such as scheduled report execution, report subscription services, snapshots, content caching, and on-demand Web access.

With these added benefits comes an additional level of management responsibility. Depending on the size of the organization, some management tasks can be delegated to other users, such as departmental managers, who might maintain report folders for their departments, as well as to system administrators and DBAs. Fortunately, SSRS provides several means of managing the report server at all levels. In this chapter, you'll continue to work with built-in tools such as Report Manager and command-line utilities, as well as with custom .NET management tools that take advantage of the SSRS programming models to administer an SSRS deployment. We'll also demonstrate SSRS's Reporting Services Configuration Manager. When connecting to an SSRS instance, it supplies much, if not all, of the functionality that is available in the Site Settings section of Report Manager and the command-line utilities. Using SSRS 2000 and 2005 once allowed some administrative tasks, like configuring report subscriptions, to be done with SSMS. However, since the 2008 release, this functionality was removed and is now only available using the Report Manager. Therefore, we have chosen to use Report Manager for most of the examples in this chapter. Report Manager can be accessed from a Web browser and does not require a local install of the SQL client tools.

Exploring Management Roles in SSRS Deployment

You can subdivide the management roles for an SSRS deployment into three basic categories:

- Content management
- Performance
- Report execution

It is always best practice to perform a test deployment of any application or service before placing it into a production environment. Because our company provides services via the Internet to a wide range of users, special management considerations—such as how to provide report subscriptions to the same report for different companies—were imperative, and we needed to test various scenarios to ensure proper functionality in each case. Let's begin by looking at the three management categories and how we implemented and tested them to ensure that when we deployed them to production, we would have few (if any) unexpected consequences. We will show how to perform all these tasks using the built-in administration tools with SSRS and SQL Server 2012. Later in the chapter, we'll show how to build a management application interface with .NET that provides the ability to schedule reports.

Managing Content

Effective management of content on the SSRS report server requires that you be familiar with the management tasks available. Several aspects of report management are available only after the report has been deployed. We'll cover each aspect in detail as we show how to deploy the health care reports for selected users:

- Shared schedules
- Report parameters and data sources
- Report snapshots, history, and caching
- Subscriptions

For each content management task, we'll provide specific real-world scenarios, continuing with the health care agencies as an example. Up to this point, you have deployed several reports, data sources, and other report items, such as graphic images and code, which you have developed throughout the book. Now it's time to put on your administrator's cap and take advantage of all the features that make SSRS a unique and powerful report-delivery system as well as one that provides a rich report-authoring environment.

Setting Up Shared Schedules

Generally speaking, a shared schedule is like a shared data source in that it is available system-wide to users who have permission to access it. You can create a shared schedule specifically for a certain job type. It is possible to configure a recurring shared schedule to execute by the hour, day, month, and week or to run only once. In this example, the financial reports will execute at the end of each month. It is important that a history be maintained for these reports so that you can freeze the values at any point in time or use them for auditing capabilities, such as understanding which users viewed a report and what they saw in the report.

You'll create a shared schedule that will be used to run the financial reports on the last day of each month to provide the following benefits:

- You can schedule the reports to execute at a predetermined time.
- You can store a snapshot of each report to maintain a historical perspective of the data.

■ **Note** Report snapshots are reports that are executed at a specific time, either when initiated by a user or as part of a schedule, and that collectively make up the report history. We'll cover snapshots in more detail later.

One financial report that customers might run at the end of the month is an AR Reconciliation report. This report lists financial transactions that occurred during the current accounting period, such as 10-2009 for October 2009. This report may be one of several that need to execute on the same schedule. Other financial reports might include an Aged Trial Balance report and an AR Aging report. We'll use the AR Reconciliation report in the following section to show the management tasks associated with setting up shared schedules and creating report snapshots. We've included the AR Reconciliation

report in the Pro_SSRS report project download, available in the Source Code/Download section of the Apress Web site (www.apress.com). The report definition file is AR Reconciliation.rdl. In this section, we'll show how to use a deployed version of the AR Reconciliation report to configure its parameter values to coincide with the scheduled times that it will execute each month.

To deploy the AR Reconciliation report, open the Pro_SSRS project in BIDS and then open the Solution Explorer. Before deploying this report, however, you need to change the project property that controls which folder the report will be deployed in on the report server. In the AR Reconciliation example, you will deploy the report to a folder called End of Month Financials. In the Solution Explorer, right-click the Pro_SSRS report project, and select Properties. Change the TargetReportFolder value from Pro_SSRS to End of Month Financials, and click OK. Now, when you right-click the AR Reconciliation report in the Solution Explorer and select Deploy, the new folder is created on your report server that contains the AR Reconciliation report.

Creating a Shared Schedule

To create the shared schedule in Report Manager, click Site Settings, and then select Schedules at the bottom of the navigation tabs on the left of the page. Click the New Schedule button link, and name the schedule End of Month Financials. In the Schedule Details section, you are presented with the standard scheduling options: Hour, Day, Week, Month, or Once. For this report, choose Month.

The first challenge, when configuring a shared schedule to run on the last day of each month, is to overcome the built-in data validation on the Scheduling form. Although it is possible to tell SSRS to execute the report on the last Sunday of every month, it is not possible to select the last day of each month because the last day is variable (that is, it could be 28, 29, 30, or 31, depending on the month). Well, it should be possible to create a single schedule to encompass all four dates, right? Not exactly! If you try to save the Shared Schedule as shown in Figure 10-1, choosing 31 causes an error when all of the months are selected because not all months have 31 days.

■ **Note** You must be logged in as a user who is a member of the SSRS 2012 System Administrator role to add new schedules. If you haven't been added to the site as a System Administrator, but you are in the local Administrators group, you may need to run your browser with elevated permissions. If using IE, you can right click the IE icon and select Run as administrator.

The solution in this example—because we know that no activity will occur after 12:00 AM on the last day of the month—is to set the schedule to run on the first day of the month at 12:01 AM. This essentially gives you the last day for every month, and you know the first day of the month will always be 1.

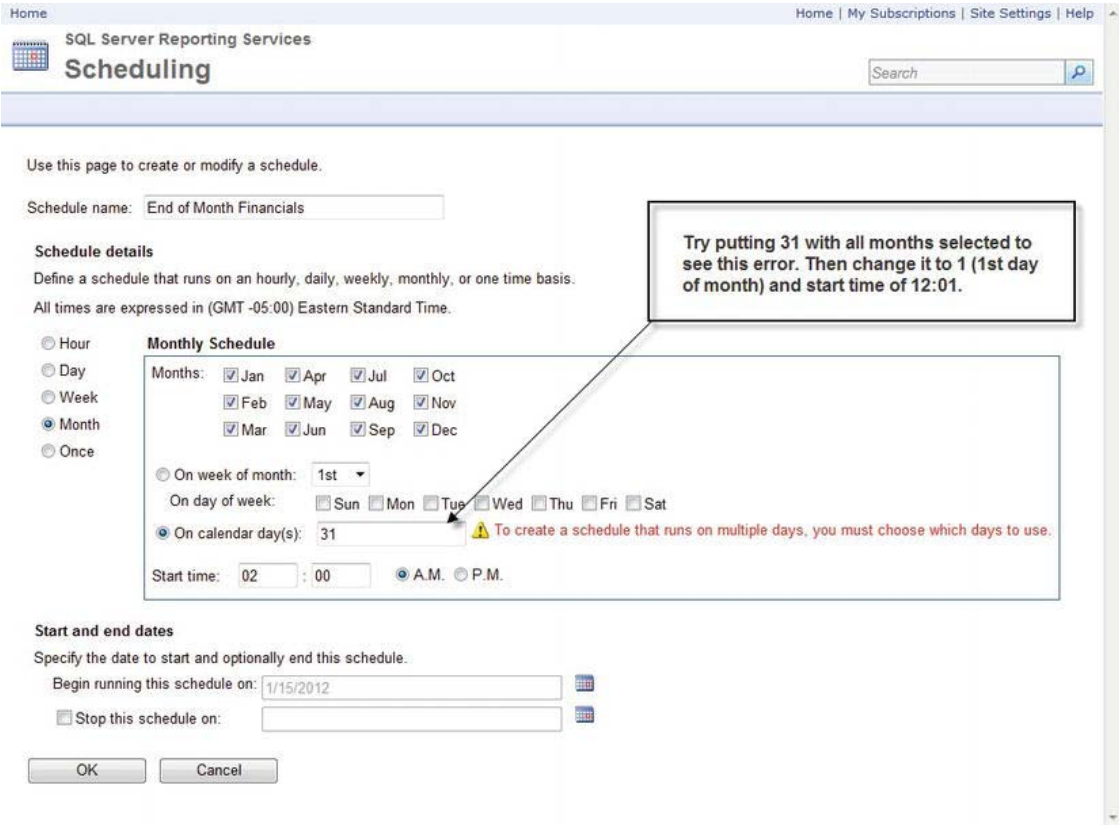


Figure 10-1. Choosing the last day of the month

Next, select an appropriate start date and end date for the schedule (in this case, don't specify an end date), and then click the OK button. Now you can move on and prepare the report itself to use the new schedule. As a note, schedules in SSRS rely on the SQL Server Agent service, and as such, the service must be running before you can successfully create a schedule.

■ **Note** Note that when a job, such as a subscription or an execution snapshot, is scheduled to run within SSRS, a SQL Server job is created using the SQL Server Agent. Jobs can be monitored through SSMS or Report Manager.

Configuring a Report to Use a Shared Schedule

The AR Reconciliation report contains report parameters that can limit the data displayed on the report. Two of the parameters, AcctPeriodYear and AcctPeriodMonth, are used in the driving stored procedure to limit financial data to an accounting period. When the AR Reconciliation report is executed from a

shared schedule, it is important that all the parameters for the report have default values. To begin with, all eight parameters in the AR Reconciliation report are set to allow NULL values, as shown in Figure 10-2, which causes the default value of the parameter to be set to NULL when executed. This is OK because the logic in the main stored procedure that uses the parameters knows to return all data when a NULL value is passed to it.

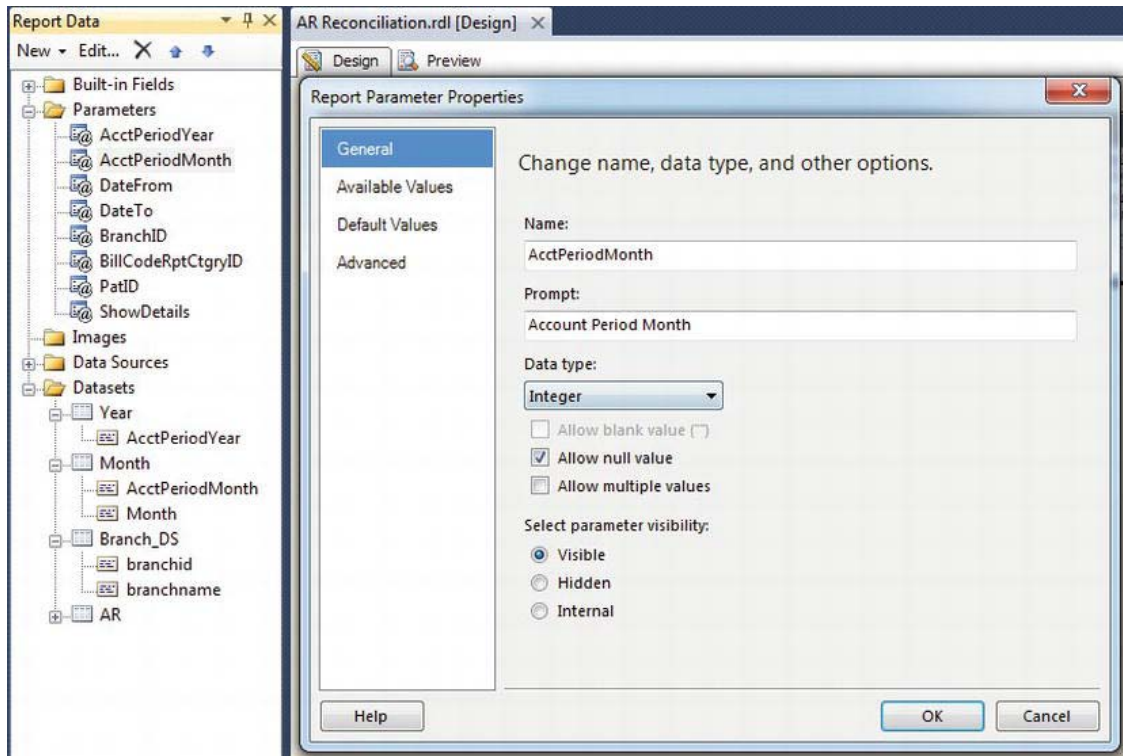


Figure 10-2. AR Reconciliation report parameters

For the parameters `AcctPeriodYear` and `AcctPeriodMonth`, however, you need to add default non-NULL parameter values so that the returned dataset includes only those records for the current accounting period. To do this, you use two functions:

- **DATEPART:** Returns an integer representing one of the component parts of a date, such as year, quarter, month, or day
- **TODAY:** Returns the current date

Used together in an expression, these functions allow you to set the desired default values for your two parameters. For `AcctPeriodYear`, you simply set the default value to the current year, as follows:

```
=DATEPART("yyyy", TODAY())
```

For `AcctPeriodMonth`, you essentially do the same thing, but you need to subtract 1 from the expression value to return the data for the correct month. For example, the expression `=DATEPART("m",`

TODAY()), when run at 12:01 AM on November 1, would return a value of 11, when the current accounting period is 10-October. To account for January, you add an IIF function to evaluate the DATEPART so that when 1 is returned for January, a 12 is returned, letting the report know to process the December accounting period. So the correct expression is as follows:

```
=IIF(DATEPART("m", TODAY()) = 1, 12, DATEPART("m", TODAY()) -1)
```

■ **Note** Our sample database contains only a snapshot of data taken from a previous point in time. As such, if you are following along the examples, you will need to enter a year and month contained within the sample database. The majority of records are between September 2009 and December 2009. However, in a real world scenario of a report requiring an End of Month frequency, use the above expressions.

Updating and Uploading the RDL File Using Report Manager

For this section, you will navigate to the AR Reconciliation report in Report Manager and modify the RDL manually to add the AcctPeriodMonth and AcctPeriodYear default parameter values via Notepad instead of using BIDS. This will demonstrate that you can make simple modifications to published reports without the need for a full-blown design environment. Open Report Manager, and navigate to the AR Reconciliation report in the End of Month Financials folder on your report server, as shown in Figure 10-3.

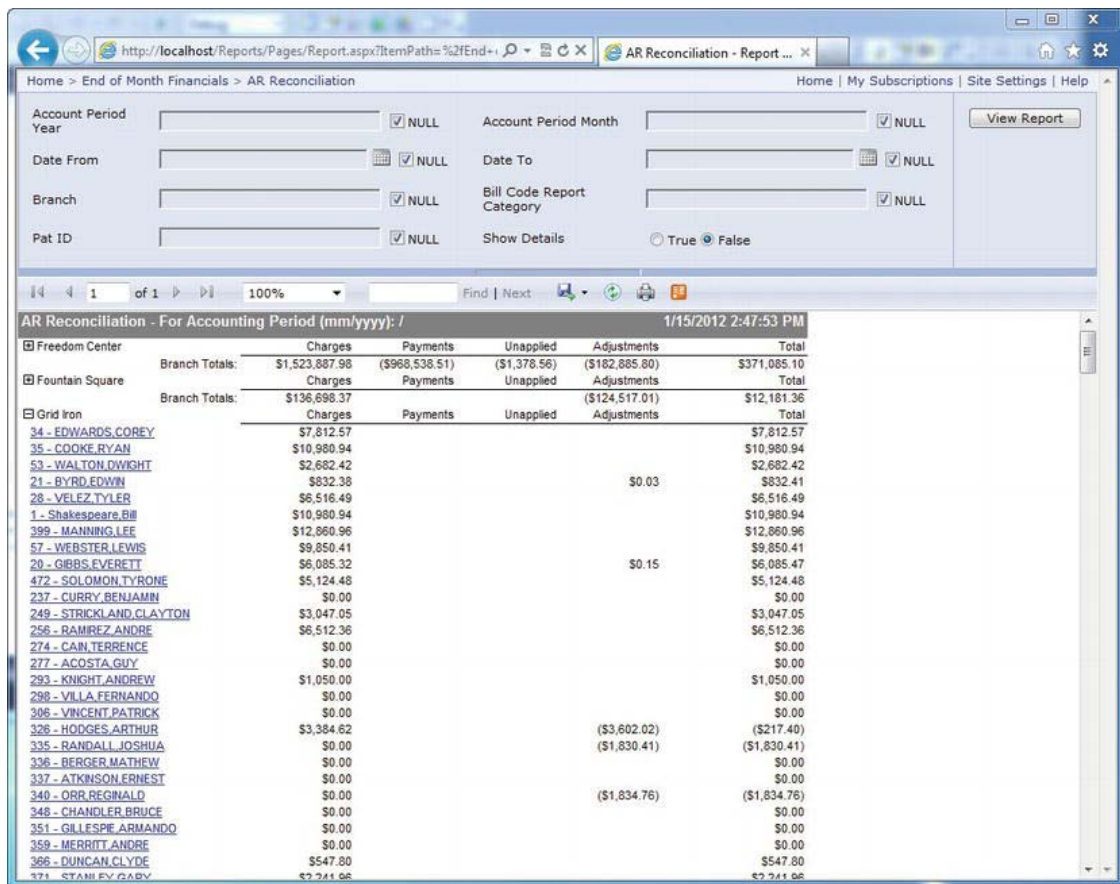


Figure 10-3. AR Reconciliation report in Report Manager

Prior to the SSRS 2008 R2 release of Reporting Services, you could get to the properties page using the Properties tab. Now, you access the report properties by placing the cursor over the report that you want to manage until you see a yellow rectangle around the report and dropdown arrow. When you see the dropdown arrow, click it and select Manage as shown in Figure 10-4. Just be sure not to click the link containing the report name as this will cause the report to execute.

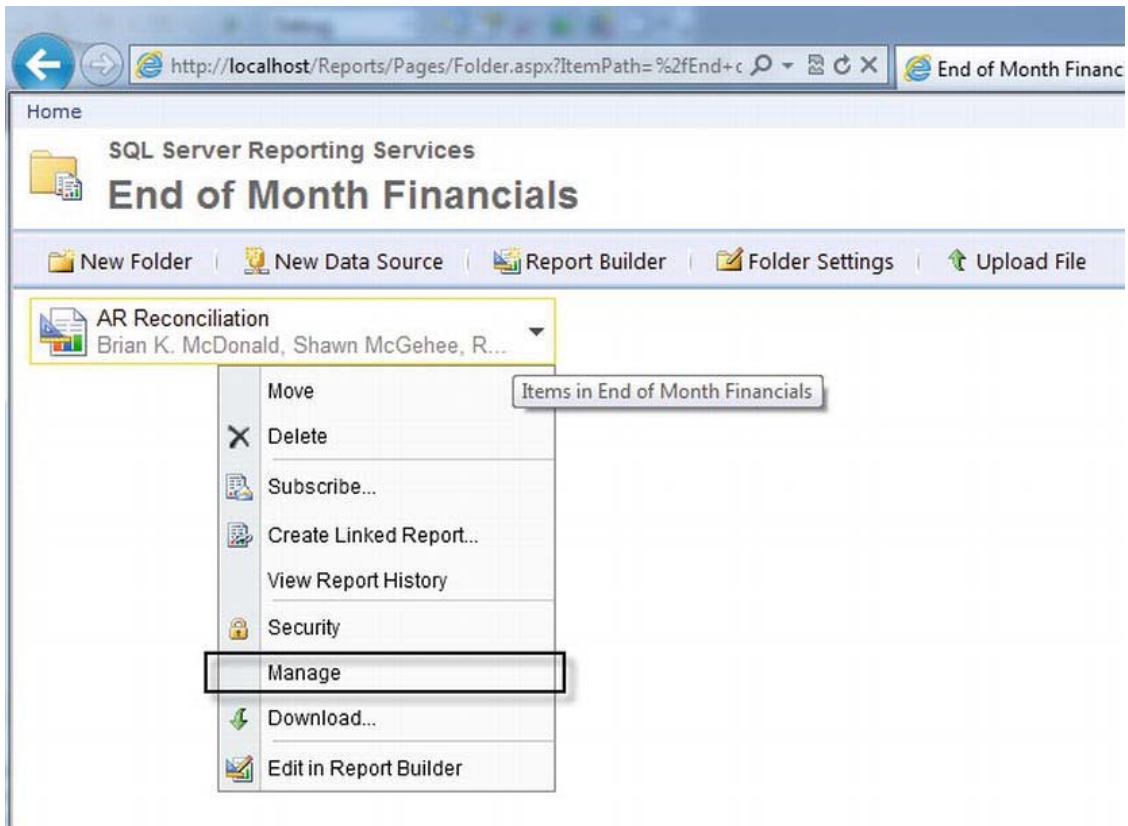


Figure 10-4. Report Management Options menu

After you have clicked *Manage*, you will be taken to the report properties page and you will see nine tabs along the left and five linked buttons on the top containing items to be used for report maintenance. The links are pretty self-explanatory, so I will not go into the details on each one. However, as you can see in Figure 10-5, you can perform such maintenance tasks as deleting, moving, downloading to edit a report, or replacing the report with another copy.

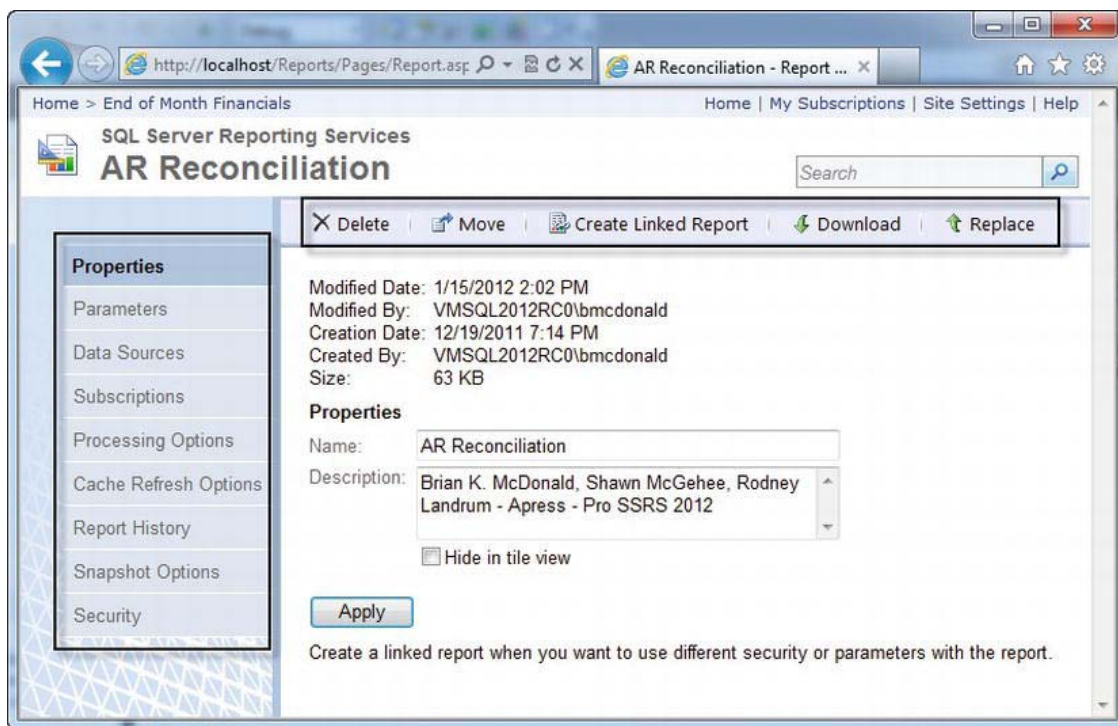


Figure 10-5. Report Properties screen

On the properties page for the AR Reconciliation report are two links—Download and Replace—that administrators can use to modify the RDL directly without having to redeploy reports through other means (such as through use of custom code, the authoring environment, or the rs command-line utility). For further details on deploying reports, please see Chapter 8. Click the Download link to save the file to a location available to you and then open it with Notepad as shown in Figure 10-6. Figure 10-6 shows the RDL file for the AR Reconciliation report. Hit the CTRL and F keys at the same time to do a find and search for ReportParameter. Click Find Next to locate the first instance of the search term entered. Notice the AcctPeriodYear and AcctPeriodMonth parameters in the RDL code.

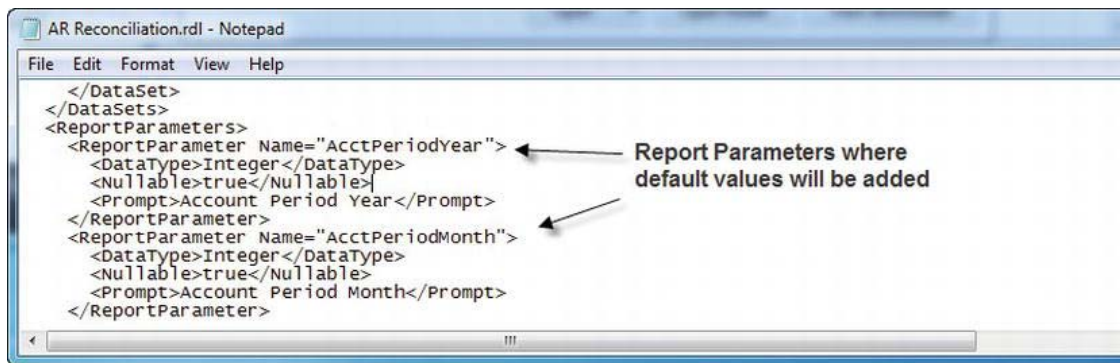


Figure 10-6. RDL file edited in Notepad

To update the report to include the default parameter values that you created in the previous section, you can place RDL code directly in the file and save it. The section of the RDL that generates the default values for each parameter is only five lines long, as you can see in Listing 10-1, which shows the default value section for the `AcctPeriodYear` parameter.

Listing 10-1. RDL Default Value Section

```

<DefaultValue>
  <Values>
    <Value>=DATEPART("yyyy", TODAY())</Value>
  </Values>
</DefaultValue>

```

Figure 10-7 shows the RDL file after the default value code was inserted for both parameters.

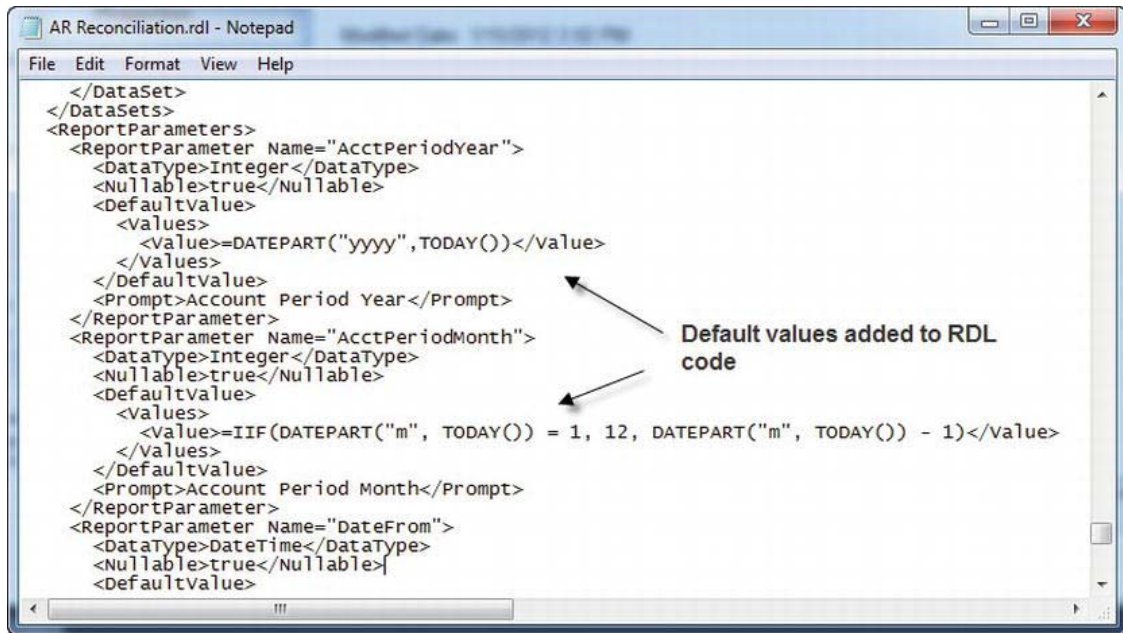


Figure 10-7. RDL file with default parameters

Because you can't save directly from Notepad to the SSRS server, save your copy of the RDL file to disk. Any accessible location is fine, such as a network share or your local disk. After the file is saved, you can click the Replace link, locate your modified RDL file on disk, and select Open. When you return back to the Upload a report definition screen, click OK to replace the existing report. If you executed the report on January 1st, 2010, you would see the report with correct default values, as shown in Figure 10-8.

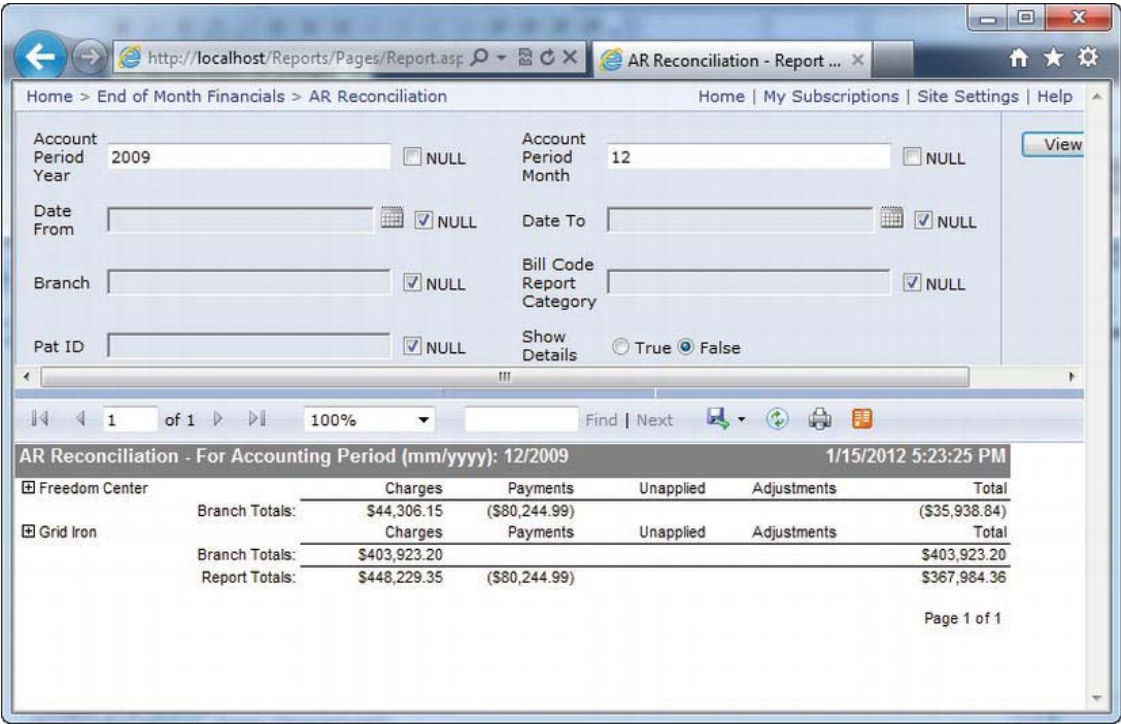


Figure 10-8. AR Reconciliation report with default parameters

Updating the report through Report Manager doesn't overwrite other properties of the report such as history, schedules, and execution methods.

■ **Caution** When you use default values based on variable data such as values in a database, it's important to note that the returned value must coincide with available values. In this example, if a value of 5 is returned for May, but no values for May can be selected, the report will force the user to make a selection, as opposed to executing and returning no data. We chose to limit the data to values actually stored in the database.

Setting Up a Data Source for the Report

The AR Reconciliation report will be set up to generate a history that allows users to view the report as it was when it was executed. In addition, because the report will be run at a prescheduled time, SSRS needs to know what credentials to use to access the data for the report. You can either modify our existing data source credentials or we could create a new data source to accommodate this.

Using the Report Manager, navigate to the folder that contains the End of the Month Financials report and then select New Data Source. Name the data source EOM Financials. Because this is a SQL

Server-based connection, you supply the appropriate connection string, which includes the server name and database or catalog for the connection. Next, choose to store the credentials securely on the server, and supply a name and password—in this case, SQL authentication credentials. Finally, choose to hide the data source in list view so that you can prevent users from accidentally selecting it when browsing. Your configuration should look similar to Figure 10-9. After you click the Apply button to create the data source, all you need to do is associate the report to the new shared data source, which you'll do in the following section.

The screenshot shows the 'New Data Source' configuration window in SQL Server Reporting Services. The browser address bar indicates the URL is `http://localhost/Reports/Pa...`. The page title is 'SQL Server Reporting Services' and the sub-header is 'New Data Source'. A search bar is present on the right.

Name: EOM Financials

Description: Subscriptions require a data source that have pre-configured authentication details. This data source will be used to execute EOM reports.

☒ Hide in tile view

☒ Enable this data source

Data source type: Microsoft SQL Server

Connection string: Data Source=(local); Initial Catalog=Pro_SSRS

Connect using:

- ☐ Credentials supplied by the user running the report
 - Display the following text to prompt user for a user name and password:

Type or enter a user name and password to access the data source
 - ☐ Use as Windows credentials when connecting to the data source
- ☒ Credentials stored securely in the report server
 - User name: AccountingUser
 - Password: [masked]
 - ☐ Use as Windows credentials when connecting to the data source
 - ☐ Impersonate the authenticated user after a connection has been made to the data source
- ☐ Windows integrated security
- ☐ Credentials are not required

Test Connection (button)

OK (button) **Cancel** (button)

Figure 10-9. Data source selections

Altering Report Data Sources

Now that we have a shared data source that we can use for our Accounting reports, we need to change the data source that the report is configured to use. We can do this from the Data Sources tab under the report Properties within Report Manager. Navigate to our End of Month Financials folder in Report Manager. Next, place your cursor over top of the AR Reconciliation report. When you see the down arrow, click it, and then select Manage followed by Data Sources tab in the left menu. We have the option here to browse to an existing shared data source or create a new one.

As one may deduce, we could have left out the previous section and performed the creation and altering at the same step. However, we wanted to show you how you could use the New Data Source button to create a new data source that could be consumed by other reports needing pre-configured authentication settings. With that being said, let's continue on and link this report to the shared data source that we just created. Click the Browse button and navigate to our EOM Financials data source under the End of Month Financials folder as shown in Figure 10-10. Click OK to alter the data source that the AR Reconciliation report was using and then click Apply to save the new settings. In the next section, we'll show you how to create snapshots to store report history.

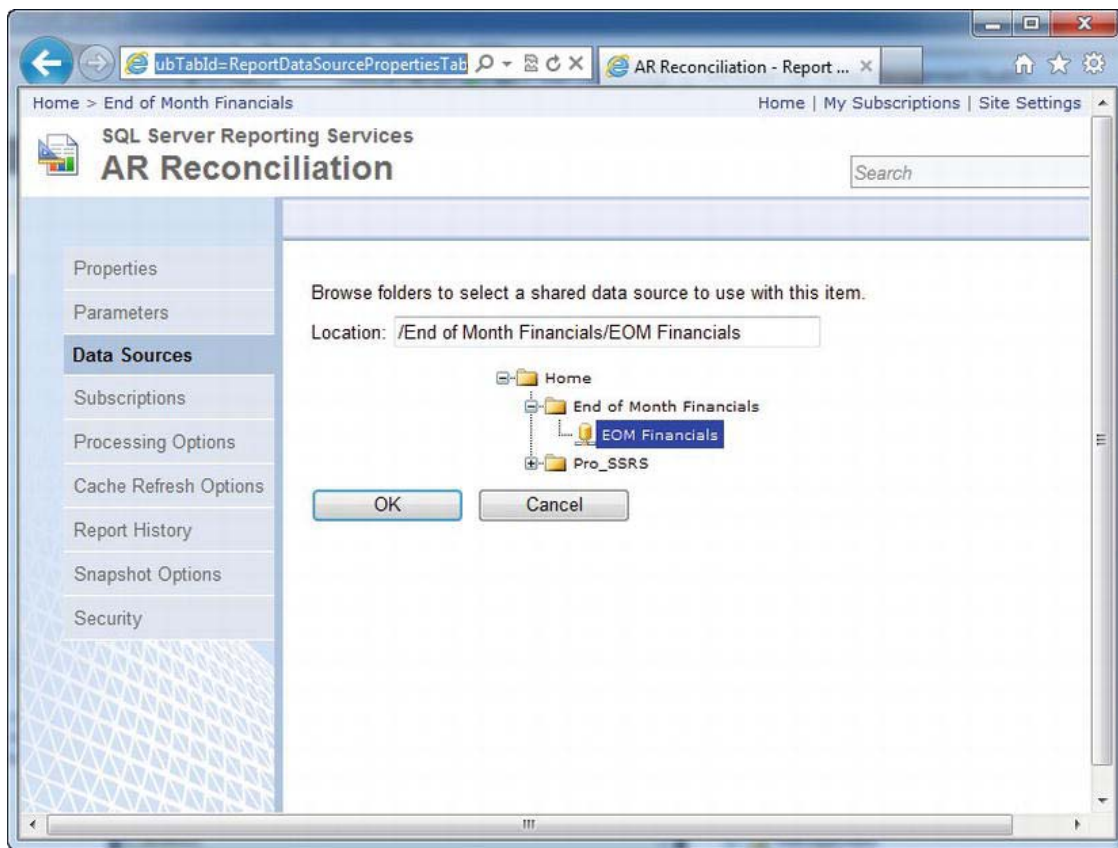


Figure 10-10. Altering a reports data source

Creating Snapshots for the Report History

The goal for the AR Reconciliation report is to allow it to execute at a specified time of month, during off-peak hours for performance benefits, and to maintain a historical picture of each month's processing. To this end, we created the End of Month Financials shared schedule earlier in the chapter. Now it's time to use a beneficial feature of SSRS, which is the capability to process a report as a snapshot.

A snapshot is a static “point-in-time” copy of a report. Two types of snapshots are used in SSRS: those generated as execution snapshots, which render a single report from a pre-executed point in time, and those generated to be stored in report history, which can contain multiple copies of a report at given points in time. In this section, we are concerned with showing how to configure the AR Reconciliation report to generate snapshots for the report history so that we can generate a series of historical financial reports.

You'll use Report Manager to configure the report history properties for the AR Reconciliation report so that a snapshot of the report is generated each time the report is processed, according to the End of Month Financials schedule.

Let's begin by looking at the available settings for storing the report history. As you can see in Figure 10-11, several settings affect not only the creation of snapshots, but also how the snapshots are stored in the report history. From within Report Manager, you can navigate to the Snapshot Options screen by placing the cursor over a report, clicking the down arrow, and selecting Manage as described earlier when navigating to the properties. Once you are in the report properties screen, click the Snapshot Options tab on the left.

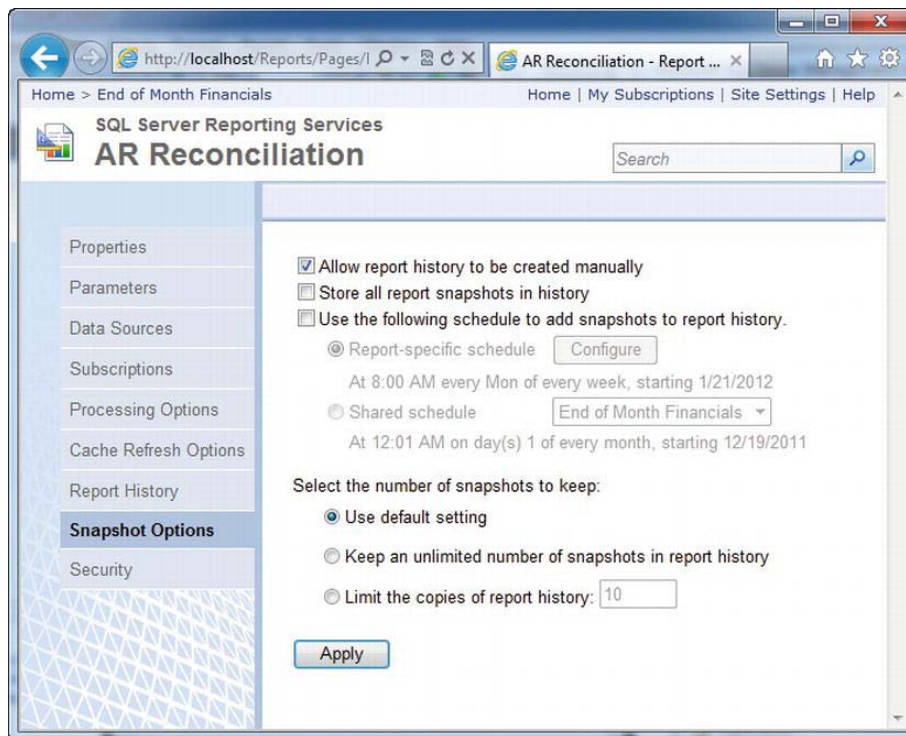


Figure 10-11. Report snapshot options

For the AR Reconciliation report, set the snapshot options so that

- Users will not be able to create snapshots for the report history. Set this entry so that only reports that are run by the schedule will appear in the report history.
- Report execution snapshots are not stored in the report history. This option is related to report execution properties, which is covered in the next section.
- The End of Month Financials shared schedule is used.
- The Default setting is used for the number of snapshots to keep in the report history. The Default setting keeps an unlimited number of reports, but you can change this via Site Settings in Report Manager. If a specific number of snapshots are selected to be kept, such as 10, then older snapshots are removed first to make room.

After making the selections as shown in Figure 10-12, click Apply.

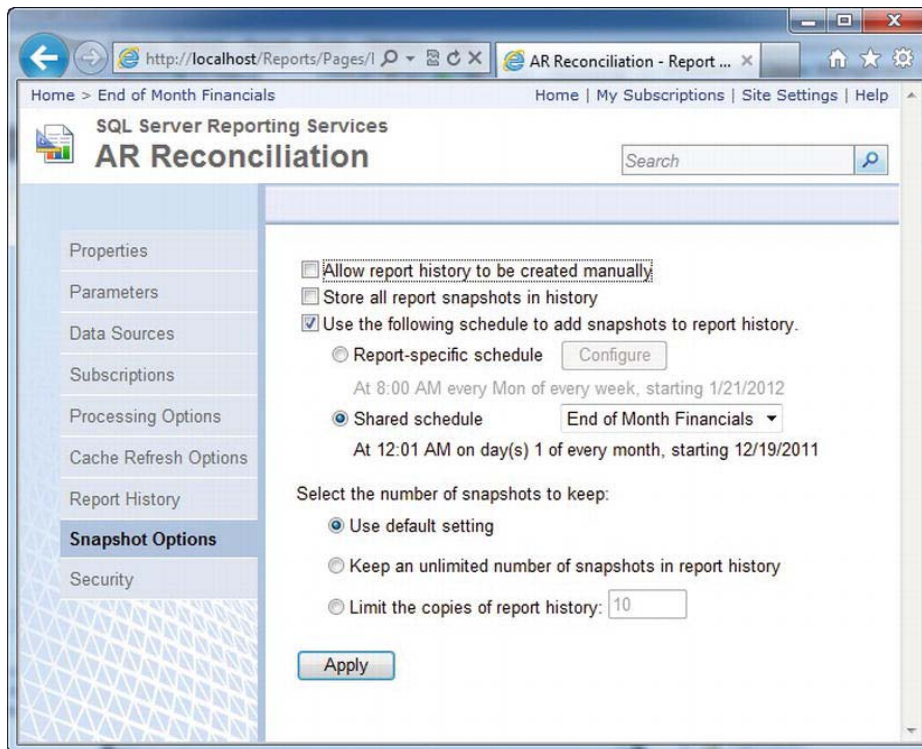


Figure 10-12. Snapshot option for the AR Reconciliation report

■ **Note** In order to create snapshots and subscriptions, you will need to ensure that SQL Server Agent is running on the server containing the ReportServer database. If it is not running, you will receive the error: “The SQL Agent service is not running. This operation requires the SQL Agent service. (rsSchedulerNotResponding)”. If you see this error, just start the SQL Agent service in the Administrative Tools or in SQL Server Management Studio (SSMS).

Over time, the snapshots will be created in the report history. Users who have access to the report history can access the snapshots through Report Manager by navigating to the History tab for the report. The History tab for the AR Reconciliation report, as shown in Figure 10-13, indicates that over a four-month period, four snapshots have been generated, as expected.

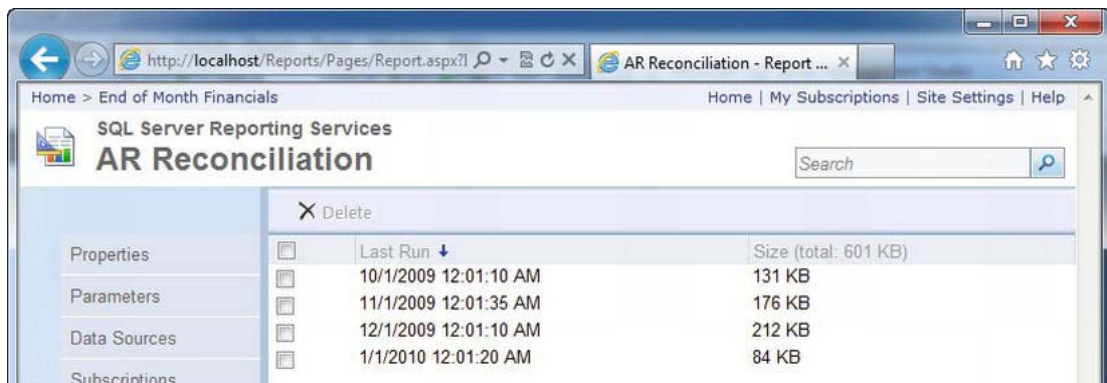


Figure 10-13. AR Reconciliation report snapshots created on schedule

It is beneficial for performance to render a report from history, first because the report has been pre-processed and second because there is no need to query the data source for the report, as both the data and layout information are stored in the snapshot. When generating large reports, such as financial reports with hundreds of pages, we highly recommend using snapshots, as well as other performance-enhancing features of SSRS such as report caching, which we will cover next.

Processing Reports and Performing Caching

The AR Reconciliation report is now set up to be delivered from a schedule, rendered from a snapshot, and saved in the report history. Employees will also run this report on-demand, meaning that users can view the report with the most recent data. Because this is potentially a resource-intensive financial report, you should ensure that performance isn't affected when the report is rendered for multiple users simultaneously. You'll use Report Manager to configure the settings that control how the report is processed.

The first step is to navigate to the Properties tab for the AR Reconciliation report and then select Processing Options in the left frame. Figure 10-14 shows the available settings for report processing.

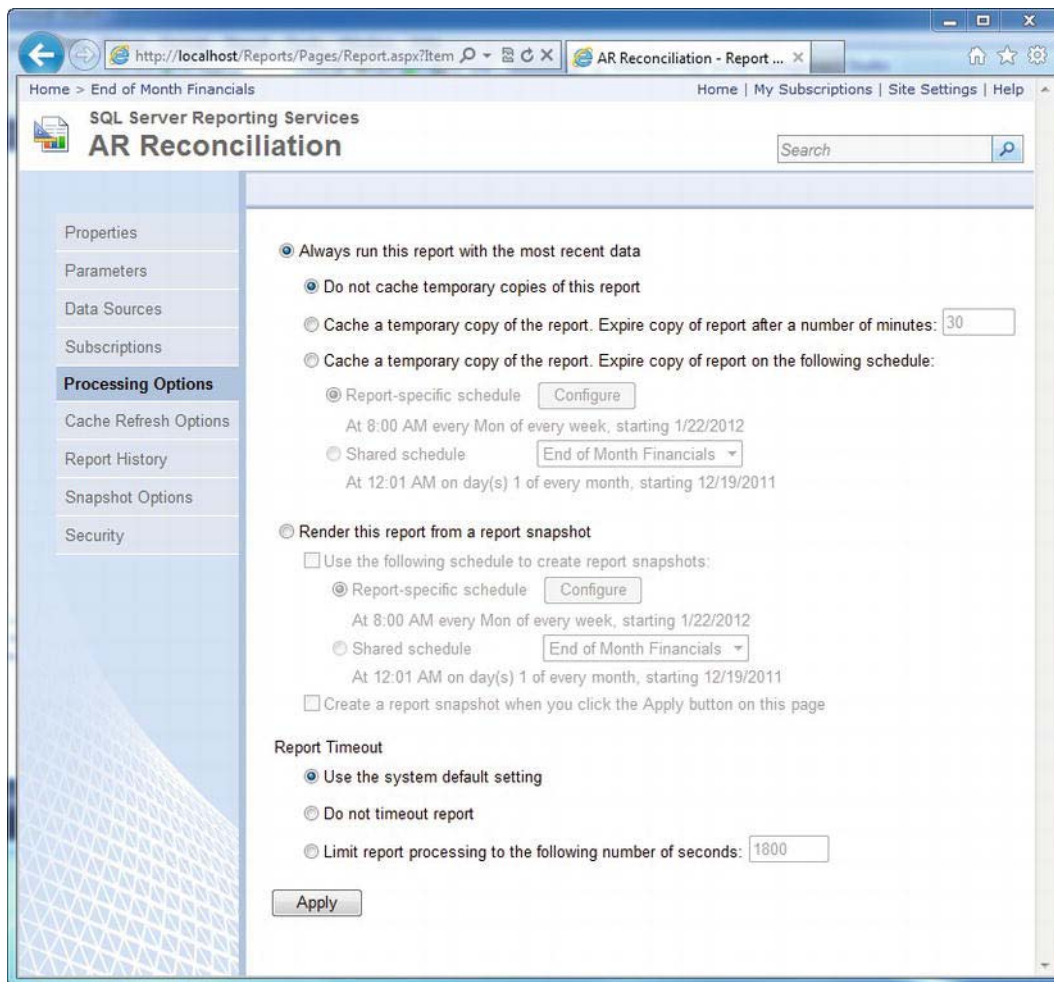


Figure 10-14. Processing options properties

The first processing selection, Always Run This Report from the Most Recent Data, has several options that control report caching.

Caching is an SSRS feature that allows temporary copies of a report to be stored and rendered to a user. The main advantage for creating a cached copy of a report is increased performance. Users who access a report that is set to be cached generate the first cached instance. Every subsequent user receives the cached copy if certain conditions are met. The conditions for a user to receive the cached copy are as follows:

- The subsequent user must access the report within the time interval before the cached report is set to expire. When the cached report expires, the next user must create a new copy.

- If the cached report has parameters that change during subsequent executions, and each user receives a new report based on that parameter, each report that's generated becomes a cached copy specific to the parameter value.
- The report's data source isn't set to Windows authentication or to prompt the user for login credentials.

For our report, instruct SSRS to cache a temporary copy of the report and expire the cached copy after ten minutes. Ten minutes is generally a good length of time to maintain cached reports, although it really depends on the time sensitivity of your data. The data in a cached report will, of course, reflect the time at which the report was rendered rather than the current time, and short of printing the execution time on the report, users have no way to know if they're viewing a cached or live report.

The second selection, *Render This Report from a Report Snapshot*, shouldn't be confused with a snapshot that creates a report history. An execution snapshot, unlike a history snapshot, is viewed from a Report folder in Report Manager just like on-demand reports would be viewed. History snapshots, on the other hand, are viewed from the Report History tab of the report and can accumulate many copies.

Snapshots don't expire like cached reports; rather, they're refreshed at a specified interval. If you choose to generate an execution snapshot for a report, then that report can't be cached. In this example, don't select this option.

The final report execution option sets the timeout interval for the report, either at a default setting, which is typically 1,800 seconds (30 minutes), or at a specified value. This is an important setting because long-running reports use valuable system resources. You'll use the default value for the AR Reconciliation report.

We mentioned the trade-off involved in using caching for reports with time-sensitive data. Another important consideration when choosing to use either snapshots or cached reports is disk space. Over time, history and cached reports set with lengthy expiration times can accumulate. The cost of disk storage compared to the performance and subsequent productivity increase is negligible, however, and shouldn't stop anyone from taking advantage of these beneficial features of SSRS.

Managing Subscriptions

Subscription services for SSRS provides a means for delivering pre-executed reports to specified locations, to a user via e-mail, to a network file share, or even directly to a printer. Using subscriptions has several key benefits. Internally in an organization, employees need key information at certain times, such as daily or at the end of a month. Externally, customers may want to receive newsletters or financial statements on a predetermined schedule. Subscriptions can accommodate both of these needs easily.

Setting up subscriptions has the added benefit of allowing you to schedule the processing of resource-intensive reports at off-peak hours, thus ensuring little or no degradation to network performance during periods of heavy usage. You'll be working with two types of subscriptions in this section:

- *Standard subscriptions*: Statically set up for one or more users.
- *Data-driven subscriptions*: Subscriber lists can be derived from multiple data source locations and can be generated from a custom query.

■ **Note** Data-driven subscriptions are by far the most powerful form of subscriptions. They're available only in the Enterprise edition of SSRS.

Managing Standard Subscriptions

You'll begin by setting up a standard e-mail subscription for employees in a health care organization that provides home-care services to patients. The report, called Patient Recertification Listing, was designed for employees who are responsible for tracking patient documentation. You can create standard e-mail subscriptions with any report. In our business, it is a requirement that the patient's documentation, in this case an HCFA 485, be completed and signed by the attending physician. The report is essentially a daily work list for these employees, where any documentation that is unsigned becomes a work item.

You can find the Patient Recertification Listing report in the Pro_SSRS report project provided in the Source Code/Download area on the Apress Web site (www.apress.com). You can deploy the report to your report server using BIDS. Open the Pro_SSRS project, and change the TargetReportFolder for the project to Patient Documentation. Next, right-click the Patient Recertification Listing report, select Deploy to create the Patient Documentation folder, and publish the report and data source to your report server. It is important to note that the Pro_SSRS data source that is provided in the Pro_SSRS project uses Windows authentication, and for subscriptions to work successfully, the credentials will need to be stored securely in the database. To change the deployed Pro_SSRS data source from Windows authentication to stored credentials, navigate in Report Manager to select the Pro_SSRS data source in the Patient Documentation folder. Change the Connect Using option to Credentials Stored Securely in the Report Server, and supply the appropriate credentials.

Because this report needed to be generic enough for on-demand viewing in addition to being used for subscriptions, we added a report parameter called Unsigned that works with a report filter to show patients with both signed and unsigned documentation. This report has other parameters as well, as shown in Figure 10-15. As you'll see, you will use the parameters when you generate the subscription.

Home > Patient Documentation

SQL Server Reporting Services

Patient Recertification Listing

Search

Properties

Parameters

Data Sources

Subscriptions

Processing Options

Cache Refresh Options

Report History

Snapshot Options

Security

Select the parameters that all users can change, and choose a default value for each.

Parameter Name	Data Type	Has Default	Default Value	Null	Hide	Prompt User	Display Text
BranchID	Integer	<input checked="" type="checkbox"/>	1	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Branch
DateFrom	DateTime	<input checked="" type="checkbox"/>	Override Default	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Certification Ending From
DateTo	DateTime	<input checked="" type="checkbox"/>	Override Default	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Certification Ending To
Unsigned	Boolean	<input type="checkbox"/>	True False	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Unsigned

Apply

Figure 10-15. Patient Recertification Listing report with parameters

Creating a Standard Subscription

The first step in manually creating a subscription is to run Report Manager and navigate to the report for which you want to create the subscription. For the example, navigate to the Patient Certification Listing report, which is in the Patient Documentation folder on the report server. Place your cursor over top of the report until you see the down arrow and click it to view the options menu. From here you have the option to select Subscribe or Manage and then navigating to the Subscriptions tab. However, this time we are going to select Subscribe to create a new subscription for the report. If you have configured your SMTP server correctly during installation, you will be given the choice to select E-mail as the delivery option. If the E-mail option is not available, you can use the Reporting Services Configuration Manager to set up e-mail, as shown in Figure 10-16. The Reporting Services Configuration Manager can set many of the same properties that control the report server as the command-line tools. The Reporting Services Configuration Manager is a good tool to use to set the properties of the report server that may need to be configured post-installation, such as the report server virtual directory or the account used to execute unattended reports. You'll use the Reporting Services Configuration Manager again to join the instance of SSRS to another report server, which creates a Web farm of report servers to gain performance. For now, verify that the SMTP settings are correct.

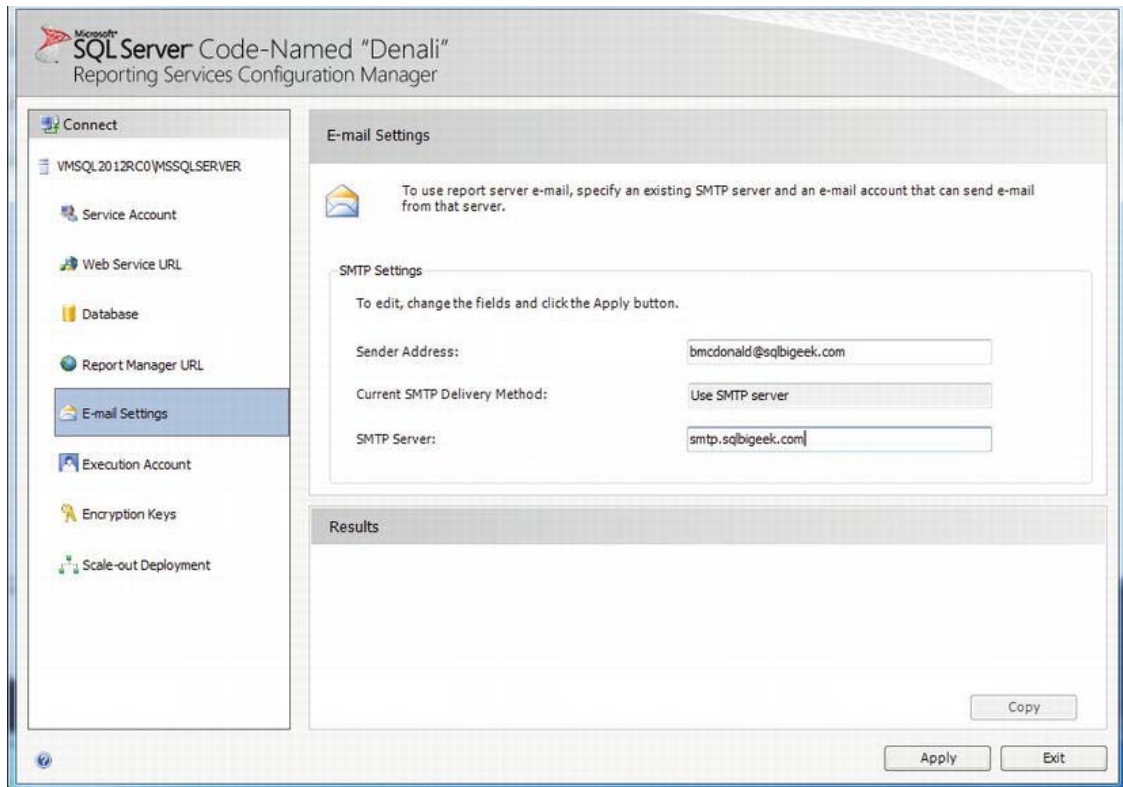


Figure 10-16. Reporting Services Configuration Manager, e-mail settings

A subscription that is delivered via e-mail provides standard delivery options for Cc, Bcc, Reply-To, Subject, Priority, and Comment, which will be the body of the e-mail message. You can also send the entire report in the e-mail, a link to the report, or both. Because the user who will be receiving this subscription will have network access to the report server, you can send just a link to the report. Sending the report itself via e-mail does have benefits, however, especially when working with users who need the report offline, such as traveling staff. We will demonstrate this in the next section when you set up a data-driven subscription. Figure 10-17 shows the e-mail options available for a report subscription.

The screenshot shows a web browser window with the URL `http://localhost/Reports/Pages/Subsc`. The page title is "Subscription: Patient Recertification Listing". The page has a navigation bar with links: Home, My Subscriptions, Site Settings, and Help. Below the navigation bar is a search box. The main content area is divided into two sections: "Report Delivery Options" and "Subscription Processing Options".

Report Delivery Options
Specify options for report delivery.

Delivered by:

To:

Cc:

Bcc:

(Use ; to separate multiple e-mail addresses.)

Reply-To:

Subject:

☒ Include Report Render Format:

☒ Include Link

Priority:

Comment:

Subscription Processing Options
Specify options for subscription processing.

Run the subscription:

☒ When the scheduled report run is complete.

At 8:00 AM every Mon of every week, starting 1/28/2012

☐ On a shared schedule:

At 12:01 AM on day(s) 1 of every month, starting 12/19/2011

Report Parameter Values
Specify the report parameter values to use with this subscription.

Branch

☐ Use Default

Figure 10-17. E-mail options for subscriptions

Configuring the Subscription

In order to provide an automated delivery system, we can configure Reporting Services subscriptions to execute at a predefined scheduled time and frequency. You can customize the schedule for individual reports or based on a shared schedule. The Patient Recertification Listing report needs to be delivered to staff members daily, and it can be run anytime after 5:00 PM on the previous day as long as it is delivered by the next business day. For our needs, a schedule of 9:30 PM every day except Saturday is sufficient, as shown in Figure 10-18, which is set by clicking the Select Schedule button under Subscription Processing Options.

Use this schedule to determine how often this report is delivered.

Schedule details

Choose whether to run the report on an hourly, daily, weekly, monthly, or one time basis.

All times are expressed in (GMT -05:00) Eastern Standard Time.

☐ Hour
☒ Day
☐ Week
☐ Month
☐ Once

Daily Schedule

☒ On the following days:
☒ Sun ☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☐ Sat
☐ Every weekday
☐ Repeat after this number of days: 1

Start time: 09 : 30 ☐ A.M. ☒ P.M.

Start and end dates

Specify the date to start and optionally end this schedule.

Begin running this schedule on: 1/28/2012

☐ Stop this schedule on:

OK Cancel

Figure 10-18. Daily schedule for Patient Certification Listing report

After making the appropriate scheduling modifications, click OK to save the changes. This report contains parameters, and we will want to configure the parameters for this subscription. As we mentioned earlier, this particular report has a parameter called Unsigned, which is a Boolean data type (either True or False) that shows both signed and unsigned documentation. The subscribers of this report, however, will be interested in seeing only unsigned documentation, so set the parameter for Unsigned to be True. Because this report has been designed to provide populated drop-downs for the parameter values that are based on individual datasets, these values are available here, such as populated Branch selections, as shown in Figure 10-19. For now, leave all the parameters, except Unsigned, with their default values.

The screenshot shows a web browser window with the address bar displaying 'http://localhost/R...'. The main content area is titled 'Subscription: Patient ...' and contains several sections for configuring report subscriptions.

Report Delivery Options
Specify options for report delivery.

Delivered by: E-Mail (dropdown)

To: Brian_K_McDonald@sqlbigEEK.com

Cc: (empty)

Bcc: (empty)

(Use (;) to separate multiple e-mail addresses.)

Reply-To: (empty)

Subject: @ReportName was executed at @ExecutionTime

☒ Include Report Render Format: MHTML (web archive) (dropdown)

☒ Include Link

Priority: Normal (dropdown)

Comment: (text area)

Subscription Processing Options
Specify options for subscription processing.

Run the subscription:

☒ When the scheduled report run is complete. Select Schedule
At 9:30 PM every Sun, Mon, Tue, Wed, Thu, Fri of every week, starting 1/28/2012

☐ On a shared schedule: End of Month Financials (dropdown)
At 12:01 AM on day(s) 1 of every month, starting 12/19/2011

Report Parameter Values
Specify the report parameter values to use with this subscription.

Branch
Freedom Center (dropdown) ☐ Use Default

Center
Freedom Center (dropdown) ☒ Use Default

Grid Iron
Grid Iron (dropdown) ☒ Use Default

Fountain Square
Fountain Square (dropdown) ☒ Use Default

Certification Ending To
1/28/2012 12:00:00 AM (text box) ☒ Use Default

Unsigned
☒ True ☐ False

At the bottom are 'OK' and 'Cancel' buttons.

Figure 10-19. Subscription parameter drop-downs

To verify that the subscription is working as anticipated, we made Brian_K_McDonald@sqlbigeek.com the sole recipient of the mail for testing. In Figure 10-20, you can see standard e-mail options for To, Cc, Bcc, and Reply-To. In the To field, you add the recipient's e-mail address and then click OK to add the subscription. After you verify the subscription's success, modify the attributes of the subscription to add the real subscribers by navigating to the report in Report Manager, selecting Subscriptions, and then selecting Edit.

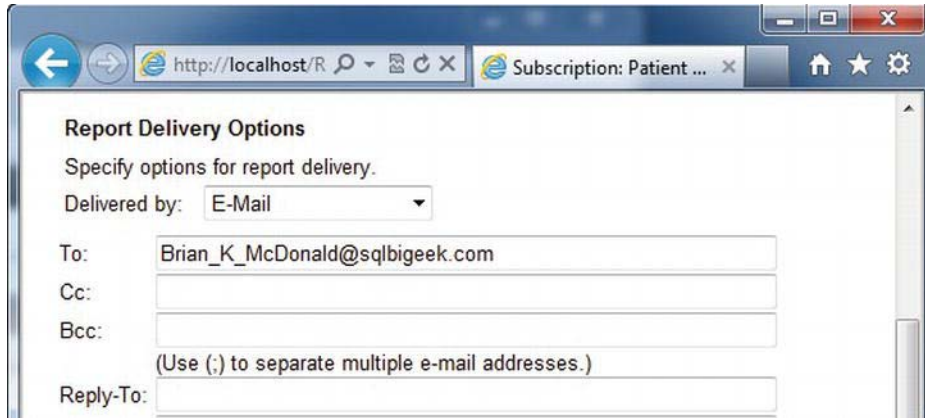


Figure 10-20. Assigning recipients to subscriptions

Managing Data-Driven Subscriptions

Standard subscriptions address the needs of many companies that want to set up custom subscriptions for both their employees and their customers. However, you can use another, much more flexible method for delivering reports: the data-driven subscription. Data-driven subscriptions allow administrators or content managers to query a data source—a SQL Server table, for example—to return a list of subscribers that meets a specific set of criteria. This is the ideal way to deliver reports to a wide-ranging list of subscribers. In addition, you have to manage only one subscription for all subscribers, and those subscribers could have different parameters that are used to generate personalized reports.

We knew we would want to let our customers and their employees take advantage of data-driven subscriptions, and fortunately we had long ago structured our application database to include employee information that would be useful for just this purpose. By storing the employees' e-mail addresses as well as other data, such as geographical locations and certifications, we had all we needed to provide a flexible delivery system, via e-mail, to traveling staff. The employees we initially targeted were clinicians who had a daily schedule of patients to see. Most of the clinicians operated laptops or PDAs as part of their daily routines.

Designing the Subscription Query

The first step was to redesign a report in SSRS so that it would provide clinical employees with their daily schedules and parameterize it so that it would be employee-specific each time it was executed. As part of the data-driven subscription, the report would be processed and delivered to employees as both an embedded, printable format such as PDF or WORD, and as a link to connect to the SSRS report server if they were online. Figure 10-21 shows the report we created, Daily Activity.

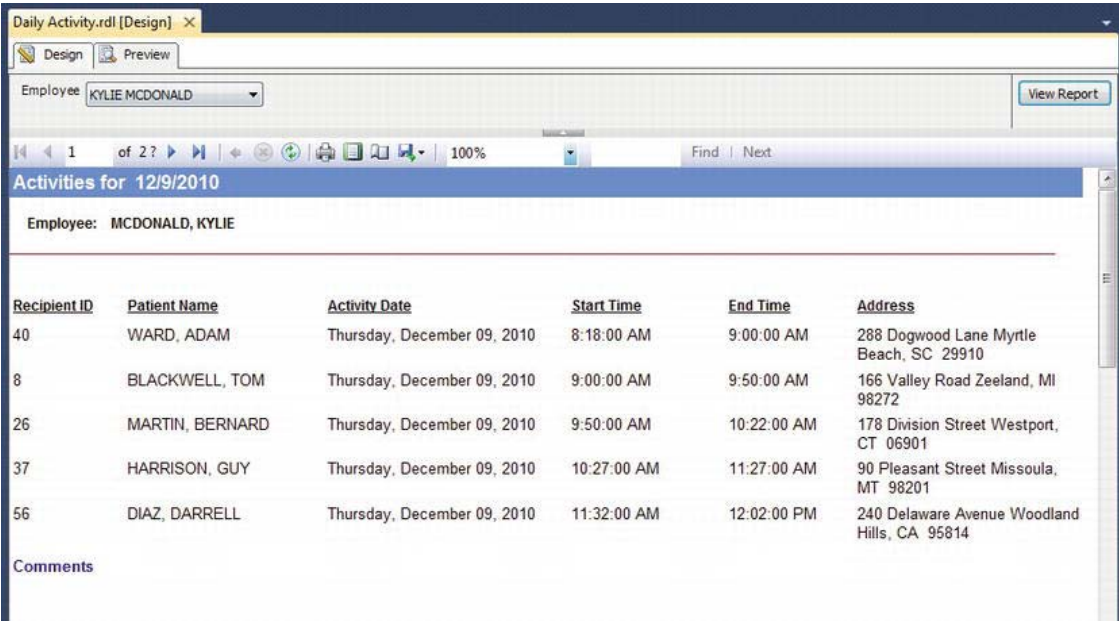


Figure 10-21. Daily Activity report

Because data-driven subscriptions are based on just that, data, a query to drive the subscription is essential. It is important to design the query to have selective criteria, because SSRS delivers a copy of the report for every record that is returned from the data source.

For our recipient list, we use the query in Listing 10-2. Essentially, the query returns all employees who have an e-mail address and who also have scheduled activities for the day following the date of report execution. The report is processed and delivered after hours. It is unusual in our environment that an employee’s schedule will change after 9:00 PM, so we set up the report to execute at that time.

Listing 10-2. T-SQL Query to Return the Subscriber List

```
SELECT DISTINCT
    EmployeeTblid, Email, HWUserLogin, ActivityDate
FROM
    Employee E (NOLOCK)
    JOIN Activity A (NOLOCK) ON E.EmployeeTblid = A.ProviderID
WHERE
    E.Email IS NOT NULL
    AND A.ActivityDate BETWEEN GETDATE() AND GETDATE() + 1
```

When this query is executed, the output of the query yields six rows of data, as you can see in Table 10-1, indicating that six clinicians have activities for the next day. You have many ways to format and compare datetime values. However, in this case, using the GETDATE function to compare the current date with the ActivityDate field value was the best choice. It was necessary to use BETWEEN with GETDATE because the ActivityDate value defaults to 00:00:00 for the time value, whereas GETDATE returns the current time. The comparison values wouldn’t match in a one-to-one comparison.

■ **Note** The data in the Activity table is static and therefore may not work as expected when you use the `GETDATE()` function, as there may be no rows to return at the time you run the query. You can easily update the activity dates to be reflective of your current datetime or change the query itself to go back (n) days where n represents the number of days to go back in time.

Table 10-1. Output of a Data-Driven Query

EmployeeTblID	E-mail	UserLogin	ActivityDate
15	NurseC@healthware.com	hwci\Nursec	2011-08-15
34	Lottah@healthware.com	hwci\Lottah	2011-08-15
44	MaryElizah@healthware.com	hwci\MaryElizah	2011-08-15
147	Fayel@healthware.com	hwci\Fayel	2011-08-15
155	Brendan1@healthware.com	hwci\brendan1	2011-08-15
159	Ethan1@healthware.com	hwci\Ethan1	2011-08-15

Creating the Data-Driven Subscription

Now we will show the procedure for creating the data-driven subscription in Report Manager. The Daily Activity report is included in the Pro_SSRS report project available in the Source Code area of the Apress Web site. You can deploy the report to any folder on your report server for testing the subscription. Make sure when you deploy the report that you modify the Pro_SSRS data source that is deployed with the report so that it stores the credentials on the report server, which is required for the subscription to be created. Open the browser, and navigate to the Daily Activity report, and from there select click the down arrow and select Manage. Once you're on the report properties page, select the Subscriptions tab on the left. On the toolbar, select New Data-Driven Subscription. Follow these steps to complete the data-driven subscription:

1. Choose a name, delivery method, and data source type.
2. Choose the data source location or define a new data source.
3. Choose the command or query to return a list of recipients.
4. Choose the settings for the E-mail delivery extension.
5. Choose the report parameters.
6. Choose when the subscription will be processed.
7. Set up a schedule for the report.

The most important of these steps is step 4, in which you specify the settings for the delivery extension. This is where you'll use the data from the driving query to instruct SSRS how to send the subscription. Every selection in step 4 has the option to retrieve the value derived from the query used in step 3, which is much more versatile than a standard subscription.

You paste the query into the query box in step 3, as you can see in Figure 10-22, and verify it by clicking the Validate button. From this point, the fields you selected in the query, namely EmployeeTblid, Email, HWUserLogin, and ActivityDate, can all be used as criteria in the remaining steps.

Home > Patient Documentation Home | My Subscriptions | Site Settings | Help

SQL Server Reporting Services

Subscription: Daily Activity

Search

Step 3 - Create a data-driven subscription: Daily Activity

Specify a command or query that returns a list of recipients and optionally returns fields used to vary delivery settings and report parameter values for each recipient:

```
SELECT DISTINCT
    EmployeeTblid, Email, HWUserLogin, ActivityDate
FROM
    Employee E (NOLOCK)
    JOIN Activity A (NOLOCK) ON E.EmployeeTblid = A.ProviderID
WHERE
    E.Email IS NOT NULL
    AND A.ActivityDate BETWEEN GETDATE() AND GETDATE() + 1
```

The delivery extension settings and report parameter values can use field values returned by the command or query. If there are field values that map to these settings, include the fields in your command or query.

The delivery extension has the following settings: TO, CC, BCC, ReplyTo, IncludeReport, RenderFor, Priority, Subject, Comment, IncludeLink, SendEmailToUserAlias

The report takes the following parameters: employeeetblid

Specify a time-out for this command: seconds

Verify that the command is correct for the selected data source:

✓ Query validated successfully.

http://localhost/Reports/Pages/Folder.aspx?ItemPath=%2fPatient+Docum...

Figure 10-22. The verified query that drives the subscription

In step 4, several settings control how the report will be sent to subscribers: To, Cc, Bcc, ReplyTo, IncludeReport, RenderFormat, Priority, Subject, Comment, and IncludeLink. For each setting, except To, you can choose a static value, a database value, or no value, as shown in Figure 10-23. You assign the To field to the Email field from the subscription query. Leave the values for all the other fields at their default settings and click the Next button to proceed to Step 5 – Create a data-driven subscription. By leaving the defaults, the subscription automatically includes the report itself and the link to the report in the e-mail to the subscribers.

The screenshot shows the 'Subscription: Daily Activity' configuration page in a web browser. The page title is 'Subscription: Daily Activity' and the URL is 'http://localhost/Reports/Pag...'. The page is divided into sections for configuring the subscription settings. The 'To' field is set to 'Get the value from the database' with a dropdown menu open showing 'Email' selected. The 'Cc' field is set to 'No value'. The 'Bcc' field is set to 'No value'. The 'Reply-To' field is set to 'No value'. The 'Include Report' field is set to 'Specify a static value: True'. The 'Render Format' field is set to 'Specify a static value: MHTML (web archive)'. The 'Priority' field is visible at the bottom.

Figure 10-23. Subscription settings

■ **Note** With data-driven subscriptions, unlike with standard subscriptions, you can control the rendering format per subscriber because it too is a data-driven setting. If you need to control the rendering format per user, you can add a field to store this value in the Employee table and select this value in the query. Also note that the default rendering format for subscriptions is a Web archive. For many types of reports, this isn't the ideal choice. Other printable reports are better suited for printed reports, such as Adobe Acrobat PDF, Microsoft Word files or static image files such as TIFF.

In step 5, you have the option to specify a static value or a value from the database. Since you want the subscription to produce one report for every record returned, you'll select the radio button to get the value from the database. The EmployeeTblid field is used for the one parameter in the report. Because you've selected this field in the query and passed this as a parameter input, each report is automatically generated with data specific to the employee who subscribed to the report. Select EmployeeTblID as the value to use as the parameter and click the Next button. The other field, HWUserLogin, is put in the driving query, which you'll ultimately compare to the Windows login name of the user executing the report. You can accomplish this, as you'll see in Chapter 11, by using the User global collection.

For the final step, you create a schedule that processes the report each weeknight at 9:30 PM, as described earlier. You can create another shared schedule to process the subscription and test it to verify that the e-mail is being delivered successfully. After that is complete, you're finished with the subscription configuration.

Performing Execution Auditing and Performance Analysis

As you deploy SSRS between development, test, and production environments, you'll find that gauging performance involves a variety of benchmarking and analysis tools. Based on the performance analysis, administrators are armed with the knowledge of what stress levels their servers can endure, and they'll be able to configure the environment accordingly. We'll show how to put the components of your SSRS deployment to the test and show how to analyze the output using standard tools.

■ **Note** The SSRS 2008 release brought us a redesigned reporting engine, with its memory management features enhanced by removing the reliance on IIS that prior versions of SSRS suffered. With that release, both Web and Windows services were combined into one Windows service.

Many agencies need to monitor and archive the details of user activity. This is especially important if you suspect there's undesired access to data. SSRS provides a built-in logging feature that captures several key pieces of information. This information is useful in two ways:

- You can capture performance information about the reports, such as the processing duration and record count.
- You can capture security information, such as who executed the report and whether or not they were successful.

The first goal in the following sections is to set up and extend the built-in logging functionality of SSRS using tools provided in the SSRS installation. You'll need to log all activity so that you can pinpoint the reports and users who are most impacting the server. We have created a custom SSRS report, Report Execution Log, which will deliver the logging statistics to administrators and contain dynamic column groupings based on a report parameter. We'll show how you can use this report for your SSRS deployment.

The second goal in the following sections is to show how to perform benchmarking tests on the SSRS servers in our test Web farm to ensure there won't be any unexpected performance problems when SSRS is deployed to a production environment. We'll show how to work with a Web application stress-test utility called Application Center Test (ACT) to gauge performance.

Configuring SSRS Logging

Getting to the execution log information in SSRS is a fairly straightforward procedure. It consists of a main table in the SSRS database called, appropriately enough, ExecutionLogStorage. When SSRS is installed by default, execution logging is enabled and set to maintain 60 days of logging. After 60 days, the log entries older than 60 days are removed from the table automatically. However, you do have the option to change the duration by connecting to the instance of SSRS using SSMS, then right clicking the server and selecting Properties. After the Server Properties windows comes up, navigate to the Logging tab. Figure 10-24 shows the settings if you wanted to change the number of days to hold 365 days rather than the default 60 days.

With the execution details being stored and being that one of the aims is to build a custom SSRS report to deliver report execution information to administrators, you'll need to be able to query the log data. Fortunately for us, Microsoft has created three views that utilize the execution details to make it easy to determine information such as the user executing the report, duration a report took to process or render and even if it was executed interactively or via a subscription. The three views in the ReportServer database are ExecutionLog, ExecutionLog2, and ExecutionLog3. We will utilize these views as a basis for our administrative report.

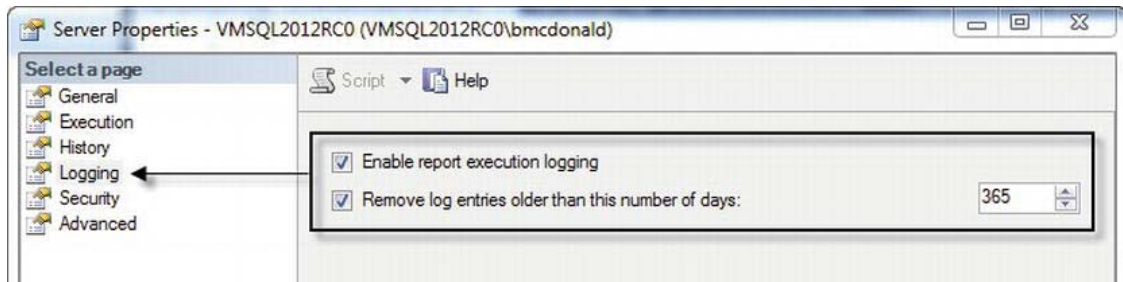


Figure 10-24. Report Server Property execution log settings

Transforming the ExecutionLog Table

Setting up SSRS to transform the logging data is a simple step-by-step procedure. Over the last several years, we have created many solutions to transfer and store the execution data into a historical database so that we do not lose any of the execution details when SSRS removes it from the database. One such solution is to use a SQL Server Integration Services (SSIS) package to transfer the execution history on a daily basis. As part of the Pro_SSRS solution, we have included a SSIS project with a single package called Pro_SSRS_Load_ExecutionLog.dtsx. Furthermore, as part of the download is the Pro_SSRSExecutionLog

database. You will need to have this database restored in order for the Pro_SSRS_Load_ExecutionLog package and the reports created in this chapter to run. See the ReadMe.txt file for detailed instructions on how to restore the Pro_SSRSExecutionLog database. You can download the Pro_SSRS project from the Source Code/Download section of the Apress Web site (www.apress.com).

The Pro_SSRS_Load_ExecutionLog package is pretty straightforward. At the top of the package, we just check for the existence of the database and if it does not exist, it branches to a statement that creates the database, a table for archiving the Execution log and loads it with all records up to yesterday. If the database already exists and the records are not already present in the database, then all of those records are pulled into the archival database from the ReportServer database. Upon scheduling this package on a nightly basis, it will get the data for the prior day. Figure 10-25 shows the package results after completing a day's load when run interactively in the development environment.

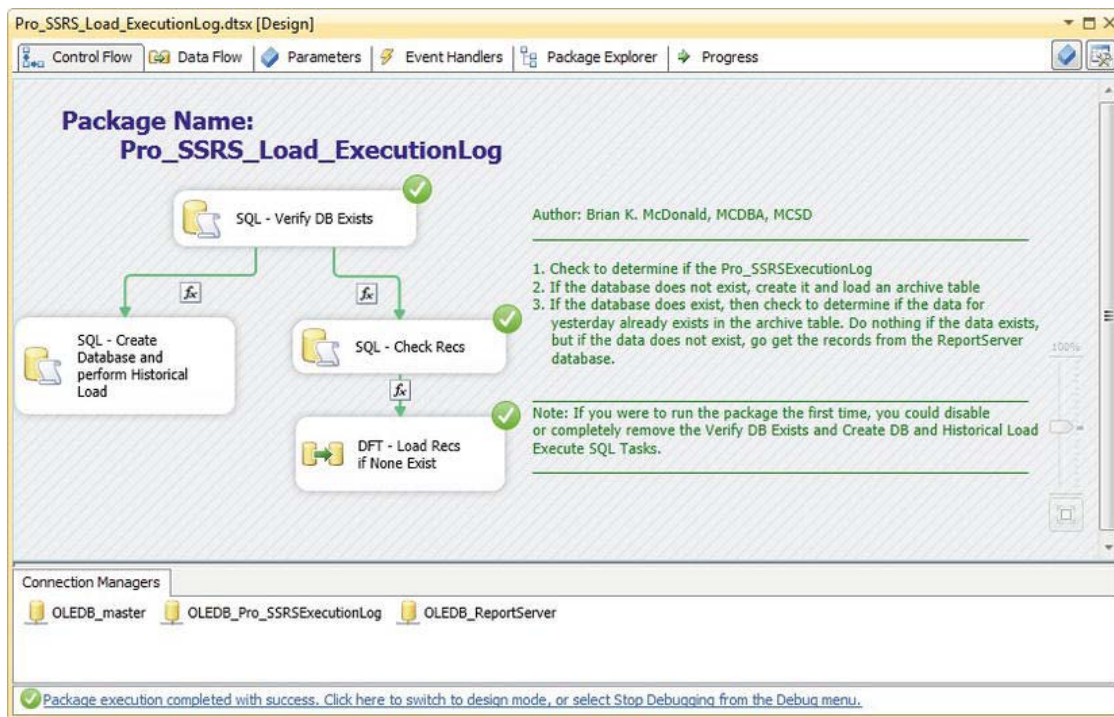


Figure 10-25. Pro_SSRS_Load_ExecutionLog SSIS package

■ **Tip** Note that the package needs to be executed regularly to keep the transformed log data current. In our situation, we created a scheduled job for this purpose that runs the SSIS package every evening.

Microsoft provides a set of sample reports that you can use with the ExecutionLog database. These reports are included in the SQL Server 2008 installation in the following location: Samples\Reporting Services\Report Samples\Server Management Sample Reports\Execution Log Sample Reports. The

sample reports are useful for giving administrators information, such as report execution by user and report size, among other things. You will find three SSRS execution log sample reports in all. One of these, Execution Summary, is shown in Figure 10-26.

In prior releases of SSRS, Microsoft has provided a set of sample reports that you can use to report data from the ReportServer database in conjunction with an SSIS package that can be used to transfer the data. However, as of today, no new version has been created for the 2012 release. As such, we have created a sample Execution Summary report that gives us a few meaningful metrics that our administrators want to keep an eye on. Some of those metrics include total number of executions, successful executions, failed executions, and number of executions by day number of the month.

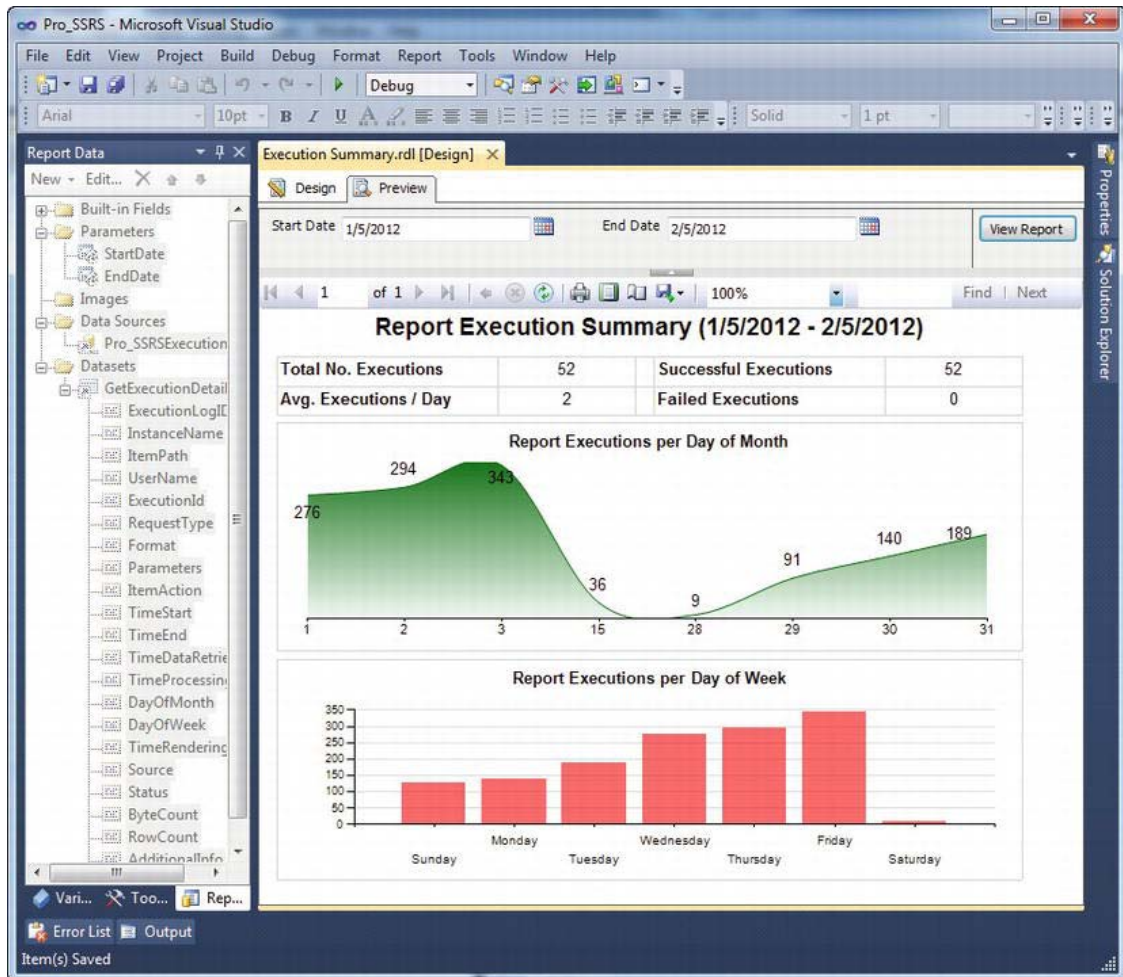


Figure 10-26. Execution Summary sample report

Designing the Log Report

We knew we would need another report that contained all the execution log information and was easy for administrators to analyze. Thus, we created a matrix-style report, called Report Execution Log, using data from a single query.

To measure performance from information contained in the execution log, you need several statistics:

- *Total time to retrieve the data:* How long did it take to retrieve data?
- *Total time to process:* How long did the report take to process?
- *Total time to render:* How long did the report take to render?
- *Byte count:* How many bytes are in the report?
- *Row count:* How many rows of data are in the report?

In addition, knowing when the report was executed is also useful. For the row groupings in the matrix, we want to see what report was executed, who ran the report, and from which client machine the report was run. For the column group, we want to have two possible selections: rendering format, such as HTML 4.0, Word or PDF, or source types, meaning how SSRS generated the report (live, cache, or snapshot, for example). `Source_Type` is an important field to monitor because how SSRS generates reports directly impacts performance. Generating reports from a cached copy or a snapshot, which are both pre-processed copies of reports, is a performance benefit. If SSRS is always generating live or on-demand reports for users, performance may suffer.

To accomplish the dynamic column groupings in the matrix, you'll use a parameter called `Column_Group` that takes the values of the field names in the query, `Format` or `Source_Type`. You'll use a default value of `Format` so that the report is automatically rendered when previewed. Both the column grouping and heading values will use the following expression to make the column dynamic based on the parameter:

```
=Fields(Parameter!Column_Group.Value).Value
```

When the report is rendered, as you can see in Figure 10-27, it will default to the `Format` field, but you can change it dynamically by changing the parameter drop-down selection to `Source_Type`.

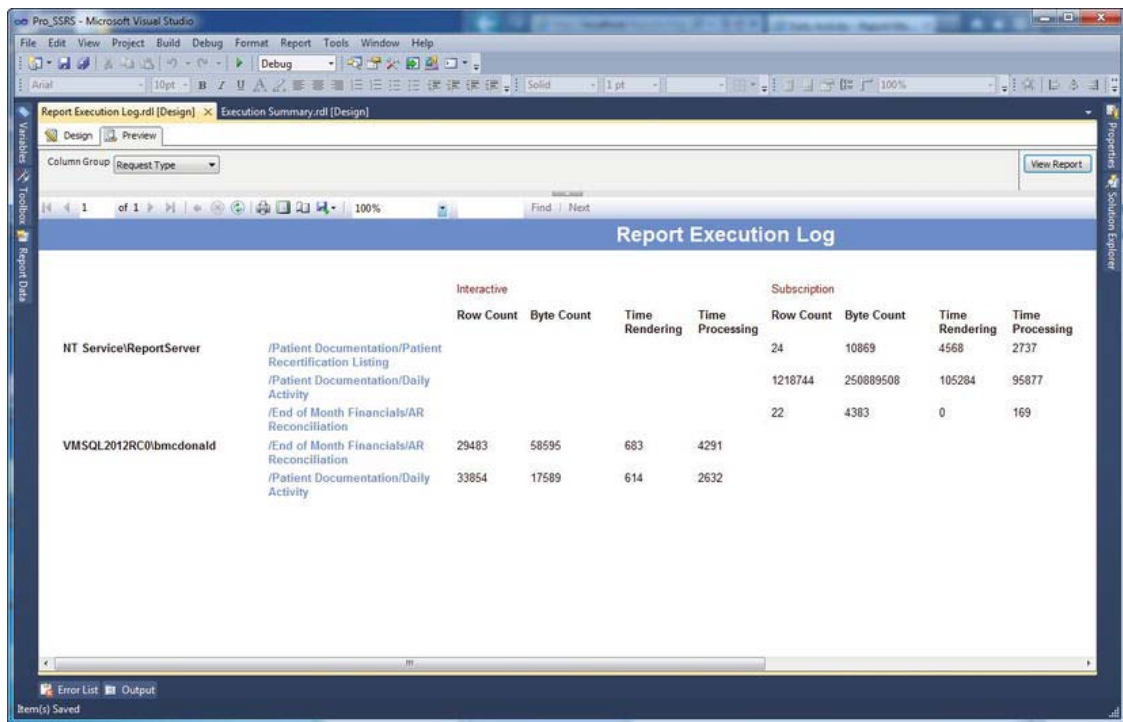


Figure 10-27. Report Execution Log report

Monitoring Performance

Of course, no one wants to experience the frustration of building a solid reporting solution in a development environment only to find out that, when deployed to the masses, it can't hold up under the strain. Generally, it's a best practice to put a simulated load on your servers to gain a better understanding of how the systems will function. As well, when you roll out a full solution, it's a common practice to roll out several pieces at a time to a limited number of users. That is what we've done in our online models.

The strategy for rolling out should also include a plan for which reports will be available on-demand versus which ones will be provided via report snapshots or subscriptions, as you've done up to this point in the chapter. Combining a strategy of peak and off-peak report processing will greatly improve performance. Another consideration for performance with SSRS lies in splitting the load of SSRS Web services and database services. That is, if the entire SSRS installation resides on the same system, this could negatively impact performance.

In this section, we'll show the results of a stress test that we ran accessing two report server instances on two separate servers, RS05 and HWC04. Many tools, such as SSRS, are available for stress testing Web applications; fortunately, the Ultimate edition of Visual Studio 2010 has a Web stress-test tool built in that we used to perform a simulated load on the two servers with up to 250 virtual users out of the box. You can find out more about the web test tool at www.microsoft.com/visualstudio.

We'll also show you how to use rsconfig to join an SSRS server to a Web farm to see how offloading resources to another system will enhance performance. The Reporting Services Configuration Manager can also be used for this purpose.

We began by running a simulated load of 15 users, all executing a single report against RS05. We quickly assessed the performance impact by monitoring the server with Task Manager. Fortunately, when running the simulated load against our Reporting Services server, it didn't take that high of a performance hit. However, in some instances you may get results that max out the CPU. In those instances, you are likely to find that the individual processes that are taxing the processor are SQL Server and the ReportingServicesService.exe. Figure 10-28 shows spikes of our CPU usage jumping between 25 and 85 percent as the test was running.

We knew our test SSRS server was a four processor system with more than 3GB of RAM. In the case where your CPU usage is getting maxed out, this tells us that it is the CPU that is the bottleneck. Because our production deployment of SSRS would not mirror the setup of RS05 (in other words, the production server would be a high-end system with at least 32GB of RAM), we could take that into consideration.

However, one other factor would have a substantial impact on the difference in performance between the production and test environments. In the test environment, the SSRS service and SQL Server were on the same system, RS05. What if we configured the SSRS service to use a remote SQL Server instance for its database? Any performance degradation caused by accessing the ReportServer database over the network instead of a local database would be negligible if the CPU utilization percentage dropped down to a more manageable number.

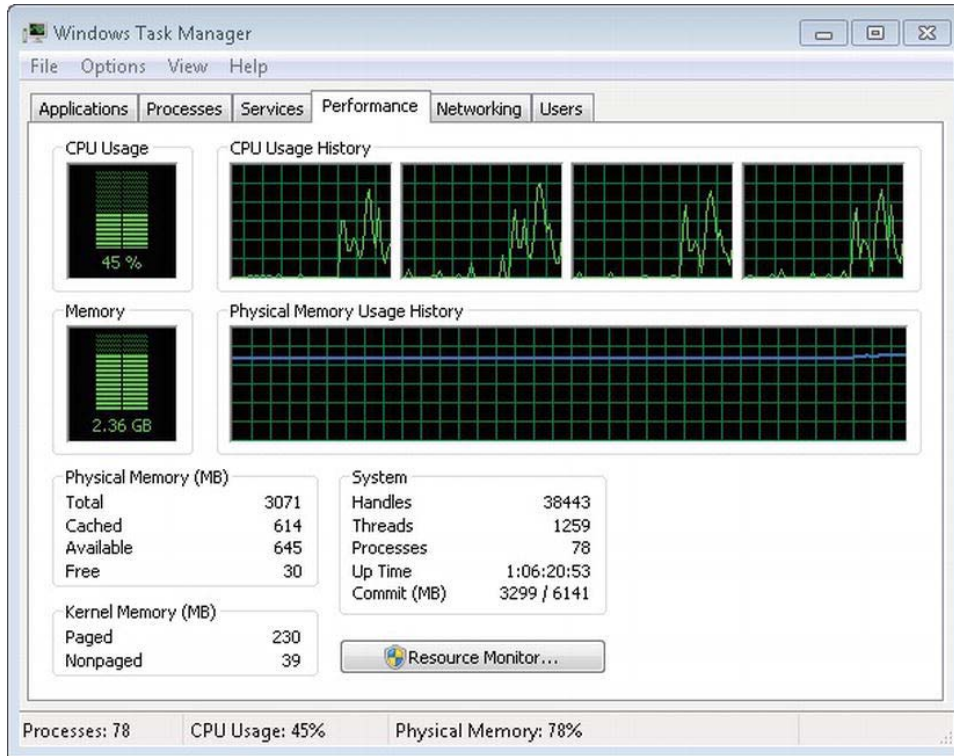


Figure 10-28. Task Manager

If you have two SSRS servers, then moving an SSRS server from one instance to another is simple. We had two SSRS servers in the test environment, RS05 and HWC04, so the move was easy enough. To instruct the SSRS service on RS05 to use the SSRS databases on HWC04, we used the command-line utility `rsconfig`. The `rsconfig` command or Reporting Services Configuration Manager is required when first joining one SSRS Web service to a Web farm that uses the same ReportServer database. The syntax for the `rsconfig` command is as follows:

```
rsconfig -c -s HWC04 -d ReportServer -a SQL -u username -p password
```

With RS05 using the remote SQL Server database, we initiated another test to see whether the CPU utilization improved. The CPU utilization improved substantially and was now under the 60 percent average.

■ **Note** Specific licensing guidelines are available for the SSRS 2012 deployment and the location of the databases and data sources. For more information, visit www.microsoft.com/sqlserver/en/us/get-sql-server/licensing.aspx.

Controlling SSRS Programmatically

There are two main methods of controlling SSRS 2012 through code:

- Web services
- Windows Management Interface (WMI)

In the following sections, we'll give an introductory look at using both of these technologies to manage reports from code. You'll use the SOAP API, otherwise known as the Report Server Web service, to add subscription functionality to the SSRS viewer that you created in Chapter 9. You'll now extend it to allow your users to add subscriptions for the reports you've developed and deployed so far. This has two main benefits. First, it allows you to offload some of the processing activity from the SQL Server and SSRS 2012 server during the day, when they are used most heavily. Second, it saves the executives who want to receive the reports from spending time navigating to the report server, entering the parameters, and waiting for the results. Each of the code listings below will be shown in C#, but VB versions are in the Pro_SSRS project download, available in the Source Code/Download section of the Apress Web site (www.apress.com).

Controlling SSRS with SOAP

The SSRS 2012 Report Server Web service offers a feature-rich way of interacting with and controlling your reporting server. Based on SOAP and operating over HTTP, the Report Server Web service is a simple, yet powerful, way to access the features of the server. In fact, SSRS 2012's Report Manager is built using ASP.NET and the SSRS Report Server Web service.

Using the Report Server Web service, you can create custom applications that control all aspects of the server and cover the entire reporting life cycle:

- Folder and resource management
- Task, role, and policy management

- Data sources and connections
- Report parameters
- Report rendering
- Report history
- Report scheduling
- Report subscriptions
- Linked reports

Adding Subscription Functionality to SSRS Viewer

You've already used the Report Server Web service to provide a list of report parameters and their possible values and to deploy reports in Chapters 8 and 9. In this chapter, you'll learn how to use the Report Server Web service to schedule reports to run automatically each morning before the office opens.

As you saw in the earlier part of this chapter, you can set up subscription services through the user interface of the report server itself. You may, however, want to provide this functionality within your customized Windows Forms (or Web) application. In the example, you'll expand on the previous Windows Forms application to allow users to provide the parameters that they want to run the report with, as well as schedule the time to run the report and indicate the delivery mechanism to use.

In the example, you'll allow the users to pick only a shared schedule that has already been defined by the systems administrator. Because you want centralized control over when scheduled reports will be run, you won't give users the ability to define their own schedules. You'll also allow them to trigger a subscription based on a snapshot. This allows them to receive their subscribed report whenever a snapshot is created for it. See the "Creating Snapshots for the Report History" section earlier in this chapter for details.

Before you run the included examples, make sure to read the ReadMe.htm file. It is located in a file in the samples root folder. If you have the code open in Visual Studio, it will be under the Solution Items folder. It contains setup and configuration steps that are required before running the examples.

Accessing an Existing Shared Schedule

If you are walking through this code, start by opening the project from Chapter 9, as it will serve as the starting point for our additions. After you have the solution, open the SSRS Viewer RVC project, and complete the following steps:

1. Select Project ► Add New Item.
2. In the Add New Item dialog box, select Windows Form, and enter PickSchedule.cs for the name.
3. With the PickSchedule.cs form open in design mode, resize it to 450×150 through the properties page.
4. Add a label, set its Text property to Schedules, add a combo box named sharedSchedules, and add a button named setSchedule and set its Text

property to OK. When you're done, you should have a form that looks like Figure 10-29.

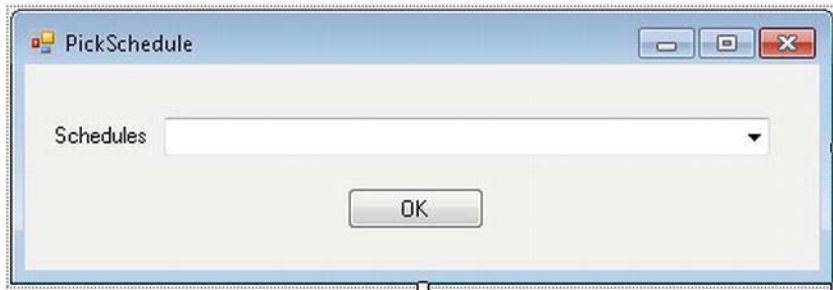


Figure 10-29. Pick Schedule dialog box

Now select View Code for the PickSchedule class. For this example, you'll add a few using statements to import types defined in other namespaces so you can avoid typing the full namespaces during the actual coding. Add the namespaces shown in Listing 10-3 to the PickSchedule.cs class file below the other using statements.

Listing 10-3. *Importing Namespaces*

```
using System.Collections;
using System.Diagnostics;
using System.Web.Services.Protocols;
using SSRS_Viewer_RVC.SSRSService;
```

Next, add the class variables shown in Listing 10-4 to PickSchedule.cs just below the class declaration. The ReportingService2010 type contains the methods and properties you can use to call the SSRS 2012 Report Server Web service and is made available through the Web reference you added to the SSRS Viewer RVC project in Chapter 9.

Listing 10-4. *Class-Level Private Variables*

```
private string url;
private string server;
private string report;
private ReportingService2010 rs;
```

Next, modify the PickSchedule_Load event to query the SSRS 2012 server for the shared schedules that are available.

■ **Note** You'll need to set up these shared schedules in advance using Report Manager on your SSRS 2008 server. You can add and edit shared schedules by navigating to your SSRS 2012 server with your Web browser, selecting Site Settings, and then under Schedules, select New Schedule. You can also use SSMS to set up shared schedules. By default, you need to set up these shared schedules as a user who is in the SSRS 2012 System Administrator role, and users who access them must be members of the System User role. In addition, subscriptions require that the SQL Server Agent is running and that your data source have stored credentials. See the "Setting Up Shared Schedules" section earlier in this chapter for details.

Next, take the URL that is passed in when the `PickSchedule` class is initialized and break it apart to get the report name for which you are setting the schedule. Add the code in Listing 10-5 to the class's constructor.

Listing 10-5. PickSchedule Constructor

```
public PickSchedule(string URL)
{
    InitializeComponent();
    url = URL;
    string[] reportInfo = url.Split('?');
    server = reportInfo[0];
    report = reportInfo[1];
}
```

■ **Note** The call to `InitializeComponent` is present in the constructor already. It was added automatically when the form was created.

To get a list of available shared schedules from your SSRS 2012 server, you'll use the `ListSchedules` method of the Report Server Web service. The `ListSchedules` method returns an array of `Schedule` objects, so after you call the method, you'll need to loop through the array to populate your combo box. Since we are expanding on the SSRS Viewer RVC project that we created in Chapter 9, we need to add two new options to our `PickSchedule_Load` event as shown in Listing 10-6. To do this, add a `Do Not Schedule` choice and a `Schedule with Snapshot` choice to your combo box.

■ **Note** You can set up snapshots through the Report Manager Web interface of your SSRS 2012 server.

The code in Listing 10-6 uses the `ComboItem` class that you created in Chapter 9 to add the items to combo boxes. With `PickSchedule.cs` in design mode, double-click the form. This creates an empty method to handle the form's Load event. Add the code shown in Listing 10-6 to the `PickSchedule_Load` method.

Listing 10-6. *Getting Shared Schedules*

```
private void PickSchedule_Load(object sender, EventArgs e)
{
    rs = new SSRSWebService.ReportingService2010();
    rs.Credentials = System.Net.CredentialCache.DefaultCredentials;
    Schedule[] schedules = null;
    try
    {
        schedules = rs.ListSchedules(null);
        if (schedules != null)
        {
            //Build list items
            ArrayList alist = new ArrayList();
            // Now add the Do Not Schedule item
            alist.Add(new ComboItem("Do not schedule", "NS"));
            // And the Snapshot schedule
            alist.Add(new ComboItem("Schedule with Snapshot", "SS"));
            foreach (Schedule s in schedules)
            {
                alist.Add(new ComboItem(s.Description, s.ScheduleID));
                Debug.WriteLine(String.Format(
                    "Desc: {0} - ID: {1}", s.Description, s.ScheduleID));
            }
            //Bind list items to combo box
            sharedSchedules.DataSource = alist;
            sharedSchedules.DisplayMember = "Display";
            sharedSchedules.ValueMember = "Value";
        }
    }
    catch (SoapException ex) { MessageBox.Show(ex.Detail.InnerXml.ToString()); }
}
```

Scheduling the Report

Now that you have the list of available scheduling options, you need to add some code to handle the case in which the user has selected to schedule the report to be delivered based on one of the shared schedules or on the creation of a snapshot. To do this, you'll use another method of the Report Service Web service, `CreateSubscription`. The `CreateSubscription` method of the API takes six parameters:

- **Report:** The full path name of the report for which to create a subscription.
- **ExtensionSettings:** A delivery extension that contains a list of settings specific to the extension. SSRS 2012 comes with two built-in extensions: the Email Delivery extension and the File Share Delivery extension.
- **Description:** A meaningful description displayed to users.

- **EventType:** The type of event that triggers the subscription. The valid values are `TimedSubscription` and `SnapshotUpdated`.
- **MatchData:** The data that is associated with the specified `EventType` parameter. This parameter is used by an event to match the subscription with an event that has fired.
- **Parameters:** An array of `ParameterValue[]` objects that contains a list of parameters for the report.

In your report scheduler, you'll create a new method, `ScheduleReport`, which is called whenever the user chooses to have a report scheduled. This method sets these parameters to the appropriate values and then calls the `CreateSubscription` method of the SSRS 2012 Report Server Web service. Most of the values are just strings and are straightforward to set.

Check to see whether the user selected a subscription and, if so, whether it is based on a shared schedule or a snapshot. You'll use this to set the `EventType` accordingly. If the user selected `Shared Schedule`, then set the variable `matchData` to the `ScheduledID`. If not, set the variable to null to tell SSRS 2012 to trigger it based on a snapshot.

```
if (sharedSchedules.SelectedValue.ToString() == "SS")
{
    eventType = "SnapshotUpdated";
    matchData = null;
}
else
{
    eventType = "TimedSubscription";
    matchData = sharedSchedules.SelectedValue.ToString();
}
```

To set up a subscription, you have to provide SSRS 2012 with some information about how to deliver the subscription. To do this, set the delivery extensions through an `ExtensionSettings` object, which itself contains `ParameterValue` objects. `ParameterValue` objects are essentially name-value pairs, making the `ExtensionSettings` object essentially an array of name-value pairs.

To use the `ExtensionSettings` object, create `ParameterValue` objects (your name-value pairs) with your delivery settings and then add them to the `ExtensionSettings` object. You'll then call the `CreateSubscription` method and pass in the `ExtensionSettings` object to give SSRS 2012 the subscription specifics. (See Listing 10-7 for details.)

If the user decides on a subscription based on a shared schedule, and the report accepts parameters, then you'll need to collect them from your report viewer interface so that you can set them in the subscription. These are the values that the report will run with whenever it's run by the subscription. To do this, you'll add code to the `PickSchedule` form to call the `GetParameters` form. Because the `GetParameters` class returns values in the form of `Winforms.ReportParameters`, you'll have to convert them into an array of `ParameterValue` objects required by the Report Server Web service. The only other item you need is the report itself, which you already have as a class-level variable that was set in the `Forms` constructor. The final method should look like Listing 10-7; add it to `PickSchedule.cs`.

Listing 10-7. Report Scheduler

```
private void ScheduleReport()
{
    // See whether the user wants to schedule this versus run it now
    if (sharedSchedules.SelectedValue.ToString() != "NS")
```

```

{
    string desc = "Send report via email";
    string eventType = String.Empty;
    string matchData = String.Empty;

    // If the user selected SnapShot, then
    // set up the parameters for a snapshot
    if (sharedSchedules.Selected.Value.ToString() == "SS")
    {
        eventType = "SnapshotUpdated";
        matchData = null;
    }
    // otherwise the user is using a subscription
    else
    {
        eventType = "TimedSubscription";
        matchData = sharedSchedules.Selected.Value.ToString();
    }

    ParameterValue[] extensionParams = new ParameterValue[8];
    extensionParams[0] = new ParameterValue();
    extensionParams[0].Name = "TO";
    extensionParams[0].Value = "someone@company.com";
    extensionParams[1] = new ParameterValue();
    extensionParams[1].Name = "ReplyTo";
    extensionParams[1].Value = "reporting@company.com";
    extensionParams[2] = new ParameterValue();
    extensionParams[2].Name = "IncludeReport";
    extensionParams[2].Value = "True";
    extensionParams[3] = new ParameterValue();
    extensionParams[3].Name = "RenderFormat";
    extensionParams[3].Value = "PDF";
    extensionParams[4] = new ParameterValue();
    extensionParams[4].Name = "Subject";
    extensionParams[4].Value = "@ReportName was executed at @ExecutionTime";
    extensionParams[5] = new ParameterValue();
    extensionParams[5].Name = "Comment";
    extensionParams[5].Value = "Here is your @ReportName report.";
    extensionParams[6] = new ParameterValue();
    extensionParams[6].Name = "Includelink";
    extensionParams[6].Value = "True";
    extensionParams[7] = new ParameterValue();
    extensionParams[7].Name = "Priority";
    extensionParams[7].Value = "NORMAL";

    // Configure the extension settings required
    // for the CreateSubscription method
    ExtensionSettings extSettings = new ExtensionSettings();
    extSettings.ParameterValues = extensionParams;
    extSettings.Extension = "Report Server Email";

    // Get the report parameters using the GetParameters form

```

```

GetParameters reportParameters = new GetParameters(url);
reportParameters.ShowDialog();
Microsoft.Reporting.WinForms.ReportParameter[] rps = reportParameters.Parameters;

// Convert the Winforms.ReportParameter returned
// from the GetParameters to ParameterValues required for
// the CreateSubscription method
int i = 0;
foreach (Microsoft.Reporting.WinForms.ReportParameter rp in rps)
{
    if (rp.Values.Count != 0) i++;
}

ParameterValue[] pvs = new ParameterValue[i];
int j = 0;
foreach (Microsoft.Reporting.WinForms.ReportParameter rp in rps)
{
    if (rp.Values.Count != 0)
    {
        pvs[j] = new ParameterValue();
        pvs[j].Name = rp.Name;
        pvs[j].Value = rp.Values[0]; j++;
    }
}

// Now set up the subscription
try
{
    rs.CreateSubscription(report, extSettings, desc, eventType, matchData, pvs);
}
catch (SoapException ex)
{
    MessageBox.Show(ex.Detail.InnerXml.ToString());
}
}
}

```

To complete the PickSchedule form, you need to wire up the setSchedule button's click event so it will call the ScheduleReport method to actually schedule the report with the schedule selected by the user. With the PickSchedule.cs in design mode, double-click the OK button. Add the code shown in Listing 10-8.

Listing 10-8. *Hooking the Schedule Button's Click Event to the ScheduleReport Method*

```

private void setSchedule_Click(object sender, EventArgs e)
{
    ScheduleReport();
}

```

Now let's add a button to the ViewerRVC.cs form that you'll code to call the new PickSchedule.cs form. First, you need to add a new button to the ViewerRVC.cs form as shown in Figure 10-30. Name it

pickSchedule, and set its Text property to Schedule. This will allow the user to pick a schedule from the viewer.

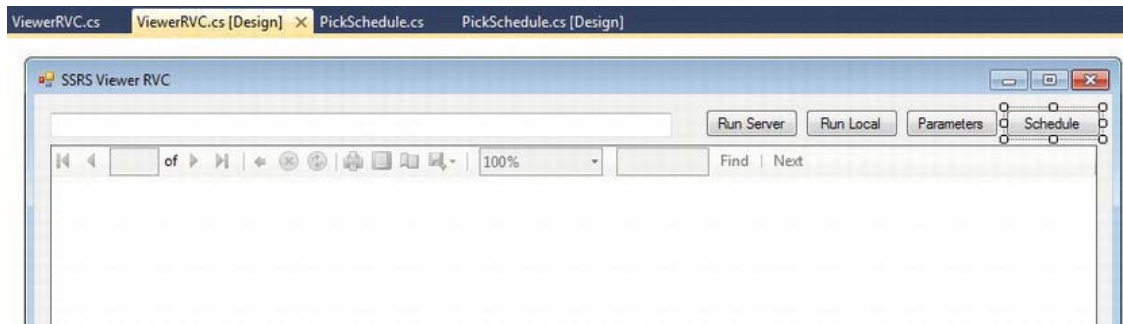


Figure 10-30. Adding Schedule button to the ViewerRVC.cs form

Second, after you add the button to the form, add the code shown in Listing 10-9 to the button's click event by double-clicking the Schedule button with the ViewerRVC.cs in design mode. Of course, you need to use the name of your report server where you see localhost in the URL.

Listing 10-9. The pickSchedule Button's click Event

```
private void pickSchedule_Click(object sender, EventArgs e)
{
    reportURL.Text = "http://localhost/reportserver?/Pro_SSRS/Chapter_7/EmployeeServiceCost";
    PickSchedule reportSchedule = new PickSchedule(reportURL.Text);
    reportSchedule.ShowDialog();
}
```

At this point, you can run the program; however, before you can schedule the report, you must set the credentials for the shared data source so the report will have login credentials to use when it is not run interactively. If you are using SQL authentication, you can do this by opening the Reports project in the solution and double-clicking the Pro_SSRS.rds data source. On the Shared Data Source dialog box, select the Credentials tab, and select Use a Specific User Name and Password. Enter the appropriate username and password to use when running this report. If you are using Windows integrated authentication, such as running reports under your Windows account, you need to use Report Manager or SSMS to edit the data source and select the Use As Windows credentials when connecting to the data source option. For Windows accounts, specify the login name using the format domain\user.

Now run the program, and pick one of your previously configured schedules. For the parameters, enter **ServiceYear 2009** and select OK to set the schedule for this report.

■ **Note** Remember to use SSMS or Report Manager to set up your shared schedules. You must be logged in as a user who is a member of the SSRS 2012 System Administrators role to add new schedules. See the section titled “Creating a Shared Schedule” earlier in this chapter for details.

Delivering the Report

In the example, you've used PDF to deliver the report to the subscription user. You've also hard-coded the e-mail address, which isn't practical in the real world. One other issue of concern, especially in the health care setting, is complying with HIPAA and protecting patient information.

You could give the user a textbox with which to enter the e-mail address to which the user wants the report delivered. However, the user could possibly type in an incorrect e-mail address and deliver the report to the wrong person. It would be great if the user's e-mail address could be filled in automatically, to make sure it is the correct address. You can do this by pulling the address from a field in a table in the database similar to the data driven subscription example given earlier in the chapter where you pulled the e-mail address from the Employee table. However, in this case, the user pulling the report may not be in the database table, and you want the report delivered automatically to the user scheduling it. Fortunately, the .NET Framework and Active Directory offer an easy way to do this. For many organizations using Microsoft Exchange Server, e-mail addresses are integrated with Active Directory. If you aren't using Exchange Server, e-mail addresses aren't integrated with Active Directory, but you can still enter them into Active Directory manually.

Let's create a method that determines the e-mail of the currently logged-in user. Then you can use it to provide the To e-mail address for the subscription. Start by adding a new reference to the project for System.DirectoryServices. Select References under the SSRS Viewer RVC project in your solution, and then select Add Reference. In the Add Reference dialog box under the .NET tab, select System.DirectoryServices from the list of component names. Next, add using statements to simplify your typing, as follows:

```
using System.DirectoryServices;
using System.Security.Principal;
```

To find the current user's e-mail address, use DirectorySearcher, which allows you to perform queries against Active Directory, as shown in Listing 10-10. You'll start at the root level of the directory and look for the user by name. When you find the user's name, you return the first e-mail address you find for the user.

Listing 10-10. Code to Query Active Directory for E-mail Addresses

```
private string GetEmailFromAD()
{
    DirectoryEntry rootEntry;
    DirectoryEntry contextEntry;
    DirectorySearcher searcher;
    SearchResult result;
    string currentUserName;
    string contextPath;
    WindowsPrincipal wp = new WindowsPrincipal(WindowsIdentity.GetCurrent());
    currentUserName = wp.Identity.Name.Split('\\')[1];
    rootEntry = new DirectoryEntry("LDAP://RootDSE");
    contextPath = rootEntry.Properties["defaultNamingContext"].Value.ToString();
    rootEntry.Dispose();
    contextEntry = new DirectoryEntry("LDAP://" + contextPath);
    searcher = new DirectorySearcher();
    searcher.SearchRoot = contextEntry;
    searcher.Filter = String.Format("&(objectCategory=person)(samAccountName={0})",
currentUserName);
    searcher.PropertiesToLoad.Add("mail");
```

```

searcher.PropertiesToLoad.Add("cn");
searcher.SearchScope = SearchScope.Subtree;
result = searcher.FindOne();
return result.Properties["mail"][0].ToString();
}

```

To use this, all you have to do is modify the `To` parameter for the delivery extension in the `ScheduleReport` method you wrote earlier to use the new method you just wrote. So, your previous code for the `To` parameter becomes this:

```

extensionParams[0] = new ParameterValue();
extensionParams[0].Name = "TO";
extensionParams[0].Value = GetEmailFromAD();

```

Now run the SSRS Viewer RVC, and choose a schedule from the shared schedules you previously configured. For the parameters, enter **ServiceYear 2009**, **ServiceMonth November**, **BranchID Grid Iron**, and **EmployeeTblID McDonald, Sherri**; this will create a subscription that is e-mailed to you on the schedule you selected. If you navigate to the server now using your browser, select the Employee Service Cost report, and then select the Subscriptions tab, you should see your subscription, as shown in Figure 10-31. If you click Edit, you see that it has provided all the parameters you selected, and it inserted the desired e-mail address in the `To` field.

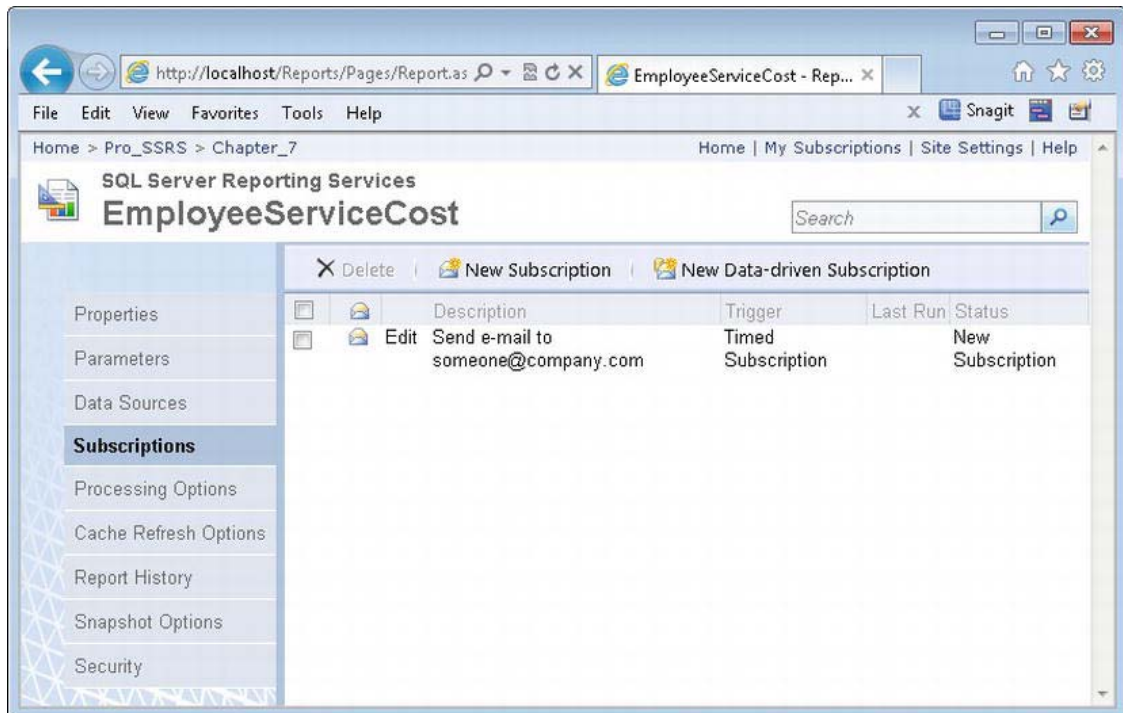


Figure 10-31. Subscription as it appears in SSRS 2012

You haven't seen all the possible options that you can use when scheduling reports such as the Employee Service Cost report, but we've given you a good start to schedule and deliver reports and add other functionality. Some possibilities include the following:

- Allowing the user to decide the format the report will be delivered in
- Allowing the user to attach the report or just provide a link
- Allowing the user to create schedules on the fly

You can use the SSRS 2012 Report Server Web service to control many more aspects of the report server and the reports under its control. We have just scratched the surface here of what you can do, but be aware that the basic aspects of dealing with the report server through the Report Server Web service are the same for nearly all the functions.

Controlling SSRS with WMI

Before we finish this chapter, we should also briefly discuss how you can manage SSRS using two WMI classes. These classes are used more for administrative tasks and allow you to access server settings programmatically. WMI is not used for manipulating reports or report settings.

WMI offers a standardized way to monitor and control systems and services running anywhere on your network. Using the WMI provider, you can write code that allows you to query the current settings of an SSRS 2012 server and also to change those settings through properties and methods of the classes providing these services.

Essentially, these providers allow you to change the settings of the configuration files on the server programmatically. So, as you might guess, the properties of these classes correspond almost directly to the elements within the XML files that hold SSRS 2012 configuration information.

Table 10-2 shows the two classes provided by SSRS 2012 for use with WMI.

Table 10-2. SSRS 2012 Classes for Use with WMI

Class	Controls	Configuration File
MSReportServer_ConfigurationSetting Re	port Server	RSReportServer.config
MSReportServerReportManager_ConfigurationSetting Report	Manager	RSWebApplication.config

You can use the MSReportServer_ConfigurationSetting class to determine and/or configure most of the database settings used by SSRS 2012 itself—that is, for the database that SSRS uses to store the reports, snapshots, and so on. This class doesn't control the data source connection information used in your reports, although you can set the login information that the server uses to run a report in unattended mode. You can also work with things such as the database server name, database name, and login credential information in this class. You can also use this class to configure the SSRS 2012 service instance name, path name, and virtual directory it maps to in the integrated HTTP system.

You can use the MSReportServerReportManager_ConfigurationSetting class to determine the instance name, path name, and virtual root of the SSRS 2012 Report Manager, as well as to read or set the URL of a particular instance.

To access this information through the SSRS 2012 WMI providers, you use the System.Management namespace, which provides access to WMI.

■ **Note** If more than one instance of a report server is installed, you'll need to locate the correct instance before reading and setting properties. The `PathName` property is the key property, and it uniquely identifies a particular instance.

Summary

SSRS 2012 provides many tools for management tasks, and we covered several of them in this chapter. Because SSRS 2012 is a full reporting solution, administrators may find it difficult to manage the entire site single-handedly without some level of automation or divided tasks, especially as the number of reports and other objects such as data sources, folders, and subscriptions grow. Maintaining these objects, whether it be to update the report via Report Manager or to mass deploy reports via a custom application, administrators will continually find themselves maintaining their SSRS report servers. Tools such as SSMS, Reporting Services Configuration Manager, and Report Manager go a long way to centralize the administrative tasks but do not necessarily reduce the potential rote tasks associated with managing a large installation. Fortunately, SSRS provides the flexibility to allow other professionals, department managers, and users to maintain their own reports using tools provided with SSRS 2012 or through your own custom applications. Of course, with this flexibility comes the need for tighter security. We will turn now, in the next chapter, to security and show how to make sure that you can lock down and monitor this flexible model.

Securing Reports

If a topic is currently on the minds of administrators more than security, we would be hard-pressed to name it. We all know that security threats come in many flavors and levels of severity—from the malicious pop-up Web pages to the invasive worms and viruses that wreak havoc on systems and take their toll on productivity by wasting time and resources to disgruntled employees to nefarious bots to skilled system intruders lurking around our data ready to pounce at any moment..

These threats are often anonymous scripts or executables—*automatons*—that their human creator has released into the wild. But what about the security violations from real individuals? These are not just elusive system hackers bent on destruction; they can be the overlooked disgruntled employee who left the company with a notebook full of passwords and the determination to make a point about the insecurity of the company's vital data.

Securing systems takes time and effort and sometimes, unfortunately, has a lower priority than other important daily tasks. However, if your company is affected by the regulations imposed by HIPAA and other laws like Sarbanes Oxley (SOX), meeting stringent security standards is a requirement, not just a recommended practice. Most companies have policies and procedures in place that will meet HIPAA and SOX compliance. As a roles-based application, SSRS will take advantage of the underlying authentication and network already at work in your organization. The SSRS security model has three important components:

- Data encryption
- Authentication and user access
- Report audits

The goal in this chapter is to meet the challenge of effectively setting up and testing each of these security components in your SSRS deployment. We will show how to do this through our experience using SSRS to meet security standards.

You can incorporate your SSRS projects in your business in multiple ways. You may have an intranet site where internal domain users are rendering reports, a .NET application that utilizes the web services to build and display reports, or even an outward facing SSRS site hidden behind an authentication portal.

Encrypting Data

When working with confidential data of any kind, the chief concern is that the only people who can see the data are those who need to see it and who have been specifically granted permission to see it. This is especially true of Protected Health Information (PHI) data, as defined by HIPAA, which has been a significant concern of ours as a software development company. Many other types of data also need this

level of protection, including financial, HR, and many more. We'll start with the first of the three main challenges we defined as crucial to a successful, secure deployment of SSRS; namely, data encryption.

Introducing Encryption

In today's mixed-technology networked environment, data encryption comes in many varieties. However, regardless of the technology, the encryption algorithms must meet a high standard for complexity and reliability. Fortunately, many applications provide built-in levels of encryption. SSRS natively supports encrypting the sensitive data it stores in the ReportServer database and configuration files. Companies may have the following other technologies in place that can be used in conjunction with SSRS encryption:

- *Wireless*: Uses Wireless Encryption Protocol (WEP), with shared keys to encrypt data transmitted through wireless access points.
- *HTTPS*: Uses a server certificate, generally from a trusted authority such as VeriSign, to provide encryption over Secure Sockets Layer (SSL). SSL is used when transmitting data with HTTPS instead of HTTP.
- *Terminal Services*: Uses Remote Desktop Protocol (RDP) for connecting remotely from a client workstation to a terminal server. This provides four levels of data encryption in Windows: Low, Client Compatible, High, and FIPS Compliant.
- *VPNs*: Allows accessibility to internal networks from VPN client systems. Encapsulates and encrypts Point-to-Point Tunneling Protocol (PPTP) and Layer 2 Tunneling Protocol (L2TP).
- *IPSec*: Is the standard security protocol for Transmission Control Protocol/Internet Protocol (TCP/IP) traffic. This adds several layers of security, including data encryption.

Securing Network Traffic Using SSL

In the following sections, we will show how to set up the SSRS server to use SSL. By having an SSL server certificate installed on the server, all data transferred between the client application (which can be a browser or custom application) and the report server will be encrypted. This is essential when transmitting confidential data such as PI information over the Internet. Having a certificate from a trusted authority such as VeriSign or Thawte also ensures that the registered domain name used to access the Web server has been validated and can be trusted to be from the legitimate company that it claims to be from.

Before we show how to install the certificate on the SSRS server, we will cover what data are being transmitted at the packet level to your SSRS server through HTTP requests. In this way, when you do actually install the certificate, you will be able to compare the data packets before and after installation to verify that the certificate is working as it should. To begin, we will show how to use a tool that is available for Windows: Network Monitor.

Analyzing HTTP Traffic

Network Monitor is a packet analysis utility that allows you to capture all of the data packets transferred to and from the target server and client. The version of Network Monitor that comes with Windows is unlike other network capture tools, such as the version of the same tool included in Systems

Management Server, in that it can listen to traffic that is destined only for the machine on which it is executed. Network Monitor is not installed with Windows by default, though. You can add it post-installation through the Add/Remove Programs applet. In this applet, select Add Remove Windows Components and then Management and Monitoring Tools. If you are unable to install from this point, you can always download the application from the Microsoft website. We will be using the newest version of this tool at the time of this writing, version 3.4.

On the SSRS server, we will show how to launch Network Monitor from Administrative Tools. If more than one network interface card (NIC) is installed on your machine, as in our case, make sure that you select the card on which you will be testing. Figure 11-1 shows the main screen of Network Monitor and the traffic that it is capturing on the network, including broadcasts and local packets. Network Monitor can be daunting to the uninitiated, as it was designed to be used by network administrators who have more than a cursory understanding of network protocols.

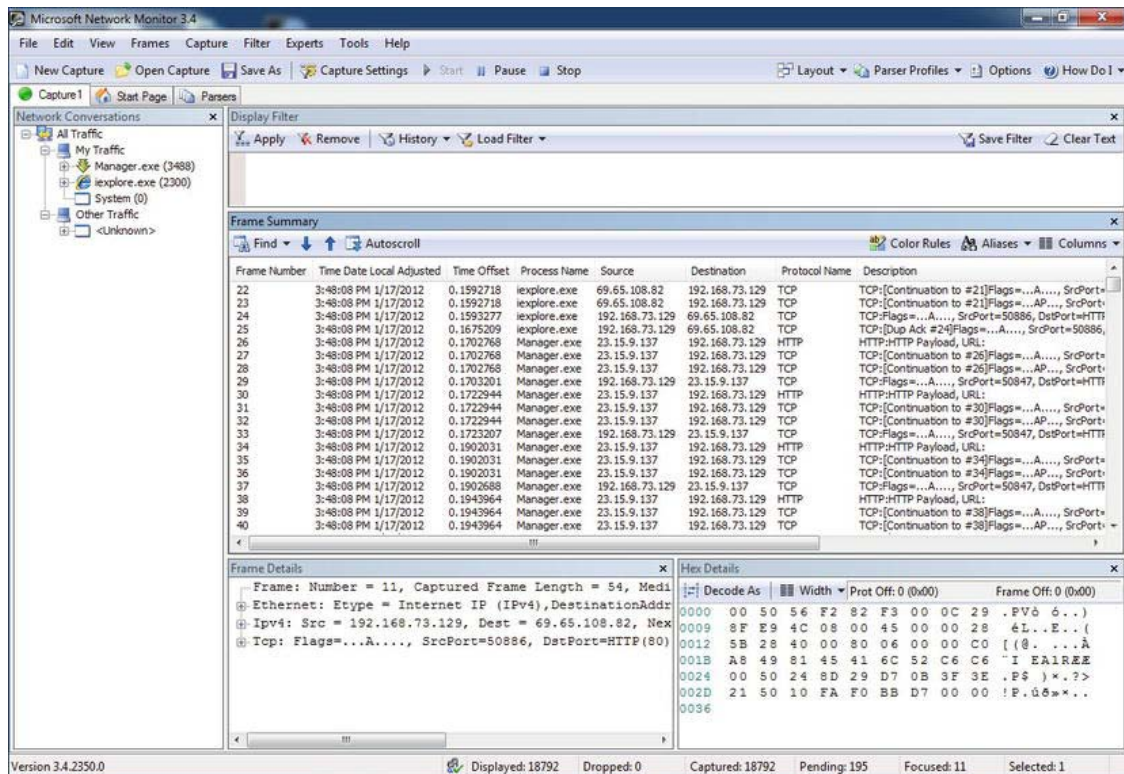


Figure 11-1. Network Monitor

You can filter out any unwanted traffic in the display, which we will cover in this section because we will be showing how to capture SSRS web data. For our example, we want to see only HTTP traffic on port 80. We will be running this directly on the client machine that accesses the report, for simplicity sake. There may be, and will be, plenty of other traffic that we don't want to see in the monitor, so we will utilize the filtering capabilities to weed out the unwanted data. You could, for example, define a capture filter that uses a pattern match in the data packet to limit the results of the capture. Alternatively, you could capture everything and then configure a display filter to limit the results. In our

case, we will setup a display filter to leave out anything that is not HTTP traffic. Setting up a filter is much easier in newer versions of network monitor than in the past. We are going to go ahead and set up a display filter so that we will only see the HTTP traffic that is flowing. To do this, start a new capture and look for the display filter window inside of the capture. If you don't see it displayed, click View->Display Filter to bring it up. It should look like Figure 11-2.

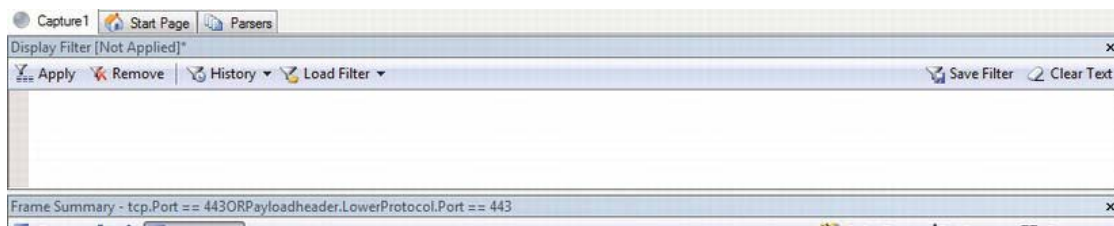


Figure 11-2. Finding the display filter window

Inside the display filter window, we need to let Network Monitor know to only show us data that are related to HTTP data. The quickest way to do this is to filter out anything that doesn't have a source or destination port of 80, the standard HTTP port. Inside of the display filter, you will set up a new rule to do just this, as shown in Figure 11-3. Click the apply button to confirm the changes.

Now, click the Start Capture button, and let the capture run as you view a page in the Report Manager. We will be taking a look at the data that are returned from this so that we can see the plain text information in HTML. We will be looking at the front page of the Report Manager to see if we can find it in the returned data for this example.

After loading the report manager, analyzing the captured frames reveals the disturbing news. You can see the HTML being returned in plain text, which is bad for us if we don't want anyone sniffing around our network to see what information could be gleaned from any of our reports. This naked HTML can be seen in Figure 11-4. The title of the page and other HTML code are plainly readable and could be used to reconstruct the entire web page or a report if it were run.

In this case, you have not analyzed other types of traffic, such as SQL requests on port 1433, to see whether other protocols are potentially sending plain-text information, but you can use the same tool to do that.



Figure 11-3. Display Filter Rule

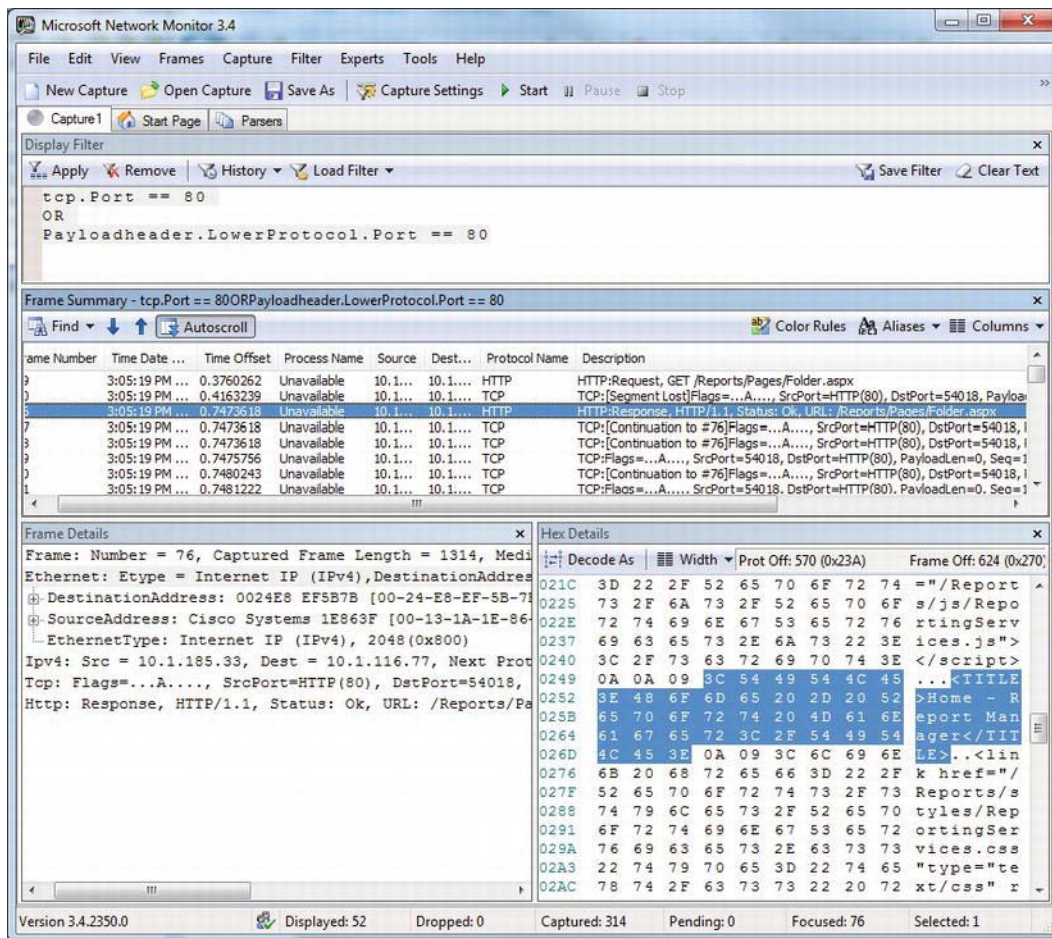


Figure 11-4. PI data captured

■ **Tip** Although we will not cover the steps to configure SQL Server itself to encrypt network traffic, as we will be doing that with the internal SSRS HTTP service, it is important to mention that SQL Server uses SSL as well, and that by having a certificate installed, you can easily configure SQL Server to transmit encrypted data. A minimal performance hit is associated with encryption. This information can be found in the APRESS book on Pro SQL Server Encryption.

Applying an SSL Certificate

Now it is time to apply a certificate to the SSRS server and rescan the traffic to make sure the viewable data in clear text will be encrypted.

Several companies provide server certificates that can be installed on a Web server and verified directly over the Internet from the trusted site that issued the certificate. By using the certificates issued from these trusted sources, such as VeriSign, the client will automatically trust the site. Other certificates, such as those generated through Certificate Services in Windows, may require that the certificate be installed on the client machine because the client will not automatically trust the certificate if it cannot reach the certificate authority. Generally, for Internet use, it is more practical to pay the fee to use the commercial certificate. The online deployment uses a server certificate issued from a commercial certificate authority. However, for the temporary test environment, you can use SelfSSL, a handy little utility that comes with the IIS Support Tools, even though we are not going to be using IIS. You can download SelfSSL from the following location:

www.microsoft.com/downloads/details.aspx?FamilyID=56fc92ee-a71a-4c73-b628-aade629c89499&displaylang=en

SelfSSL will generate and automatically apply a temporary certificate to a Web site. You run SelfSSL from the command line on the server on which you want to add the certificate. Once it is installed, you can open a command prompt for SelfSSL by clicking Start ► All Programs ► IIS Resources ► SelfSSL. The typical syntax will be in the following format:

```
Selfssl.exe /N:CN=MACHINENAMEHERE /V:20 /T
```

The /N:CN=MACHINENAMEHERE /T option indicates that the common name on the certificate will be the name of the server (You will substitute your server name in the command). The /V:20 portion indicates that the certificate is valid for 20 days. The /T option instructs SelfSSL to add the certificate to the Trusted Certificates list so that the local browser will automatically use the certificate when connecting to the site. You can manually install a local copy of the certificate on other client machines that will access this server. Because SelfSSL installs the certificate that it generates, you don't need to go through the process of generating a certificate request, which would normally be sent to a certificate authority.

■ **Note** When you create the temporary certificate on your test server, it will ask if you would like to change the settings for a website on your system. There is no need to do this since we will be binding the certificate in the configuration manager manually. If you do, you may receive an error on opening the metabase, but you can ignore this.

After we create and install the new SSL certificate, we need to configure SSRS to use this certificate to encrypt the HTTP traffic sent via Report Manager. Open the Reporting Services Configuration Manager by clicking Start ► All Programs ► Microsoft SQL Server 2012 ► Configuration Tools ► Reporting Services Configuration Manager. Once open, connect to your SSRS 2012 instance, and click the Report Manager URL link in the left-hand navigation bar.

From here, click the Advanced button to open the Advanced Multiple Web Site Configuration screen. You will notice that there are two sections in this configuration window. The top section is used to configure HTTP identities and the bottom for HTTPS/SSL configurations. Click the Add button in the

SSL section, and in the Certificate drop-down select the certificate you just created. You can also configure a different port or specify an IP address for use, but we will be using the default for both. Figure 11-5 shows you how the configuration screen should look.

After clicking OK on both open dialogs, the SSRS Server configures and binds the new URL/port combination, and the URL is then ready for use. You will now see two URLs for use in the identification section of the Reporting Services Configuration Manager.

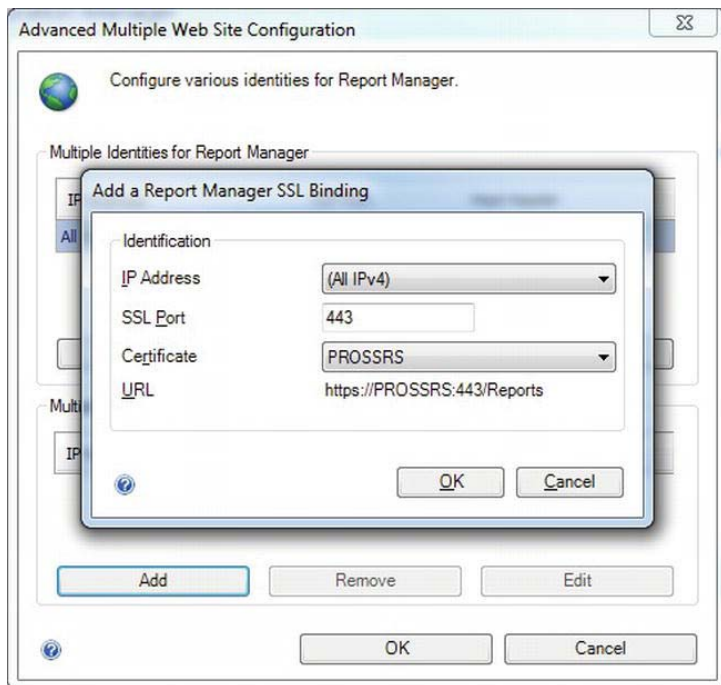


Figure 11-5. Configuring the HTTPS URL

Capturing HTTPS Traffic

Now that you have the certificate installed, let's return to Network Monitor and capture the running reports, this time using https in the URL to the report server on PROSSRS in this example, which instructs the browser to connect to the site with SSL on port 443, instead of HTTP on port 80.

The first thing you may run into when you navigate directly to the secure report manager is a warning that the certificate has not passed all the criteria to be trusted because it does not come from a known certificate authority (see Figure 11-6).

You can select Continue to This Web Site because you do indeed trust the site. You could also install the certificate on the local machine by clicking the lock at the bottom of the browser and selecting Install Certificate so that you will not be prompted with this message again. Installing the certificate in the local client's certificate store causes the browser to automatically trust the site. These steps are not required for known certificate authorities such as VeriSign but are required for this self-assigned certificate.

At this point, you can access Report Manager with either HTTP or HTTPS since we have not removed the ability to access via port 80 in the Reporting Services Configuration Manager. You can control the required level of security in a few ways.

One method to control the level of security that SSRS will use is via the service config file, `rsreportserver.config`, located in the installed folder, typically `Drive Letter:\Program Files\Microsoft SQL Server\MSSRS11.MSSQLSERVER\Reporting Services\ReportServer`. Open the file in Notepad, and look for the following entry:

```
<Add Key="SecureConnectionLevel" Value="0"/>
```

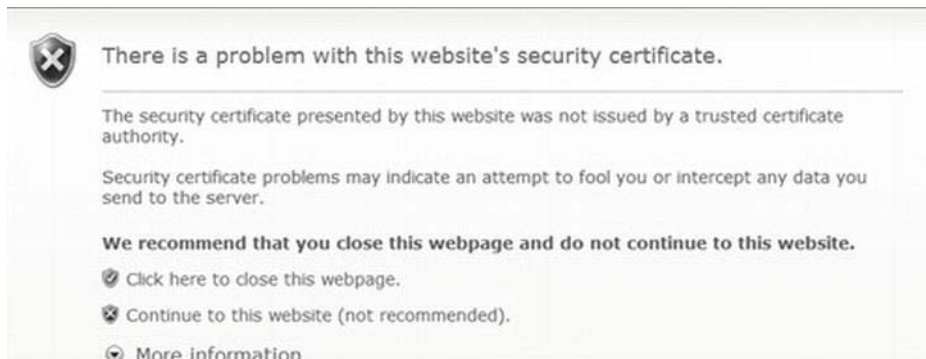


Figure 11-6. Warning for nontrusted security certificate

Four values control the level of security, 0 through 3. The default for a deployment that does not configure SSRS for SSL during installation is 0, which is the least secure. A value of 3, the most secure, requires that every SOAP API call uses SSL. For this example, set the value to 2, which will require encryption of all report data. All calls to the server will now automatically use port 443 and encrypt the data, including the URL string itself, which is important if you are passing any possibly sensitive information in the URL. If a user tried to connect to the Report Manager or report server URL using HTTP, the report server would automatically redirect the client to HTTPS to require a secure connection. You should also add the SSL certificate for use with the Report Service URL section in the SSRS configuration tool before restarting the services to make this change effective. It will need to have a valid certificate to use encrypted communication as well.

You can also remove the HTTP access through the Reporting Services Configuration Manager. In the same section that we used to add the new SSL secured address, you can also remove the HTTP address binding. This would require that any user trying to access the site would have to use the HTTPS address and have all of the Web data encrypted.

We will also need to make a small change to the display filter that we setup earlier. We are looking for port 80 traffic in that example, but we need to now be searching for port 443. Change your display filter so that each instance of port 80 is now searching for port 443 and click the apply button to confirm that change.

When you capture the frames in Network Monitor while using the new secure address, you can see that all of the previous HTTP data on port 80 are now using SSL on port 443, as shown in Figure 11-7. The data are encrypted and we can also see the SSL protocol handling the handshake setup so that all of our HTTP data is safe and secure.

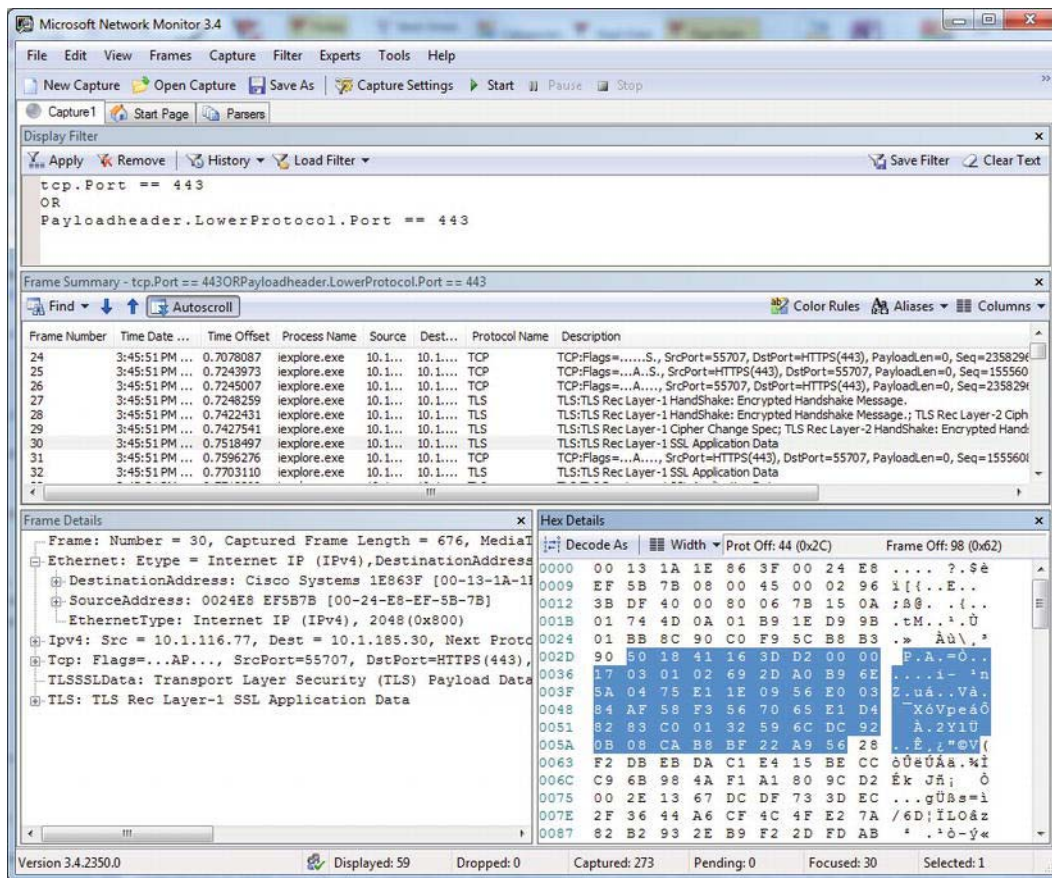


Figure 11-7. Network Monitor with encrypted packets

Securing Data Storage in SSRS

While it is important to ensure that network traffic is encrypted, this is only one aspect of maintaining a secure environment. SSRS requires that sensitive data, such as account information used for data access, be stored securely. Since these data are stored in different locations, such as database tables and configuration files, SSRS uses a symmetrical key encryption process to securely store and access this information. What this means is that the authentication information in the database and configuration files is stored in an encrypted format, and SSRS uses the encryption keys it generates to decrypt the information when needed.

As with many SSRS tasks in SQL Server, multiple tools are available to make configuration changes to the report server. You can manage keys with the Reporting Services Configuration Manager as well as a command-line utility called RSKeyMgmt. You can use either of these tools to back up the keys associated with the report server instance so that if something were to occur that caused the server to have to be rebuilt, you could reapply the keys to the installation. The encryption keys are generated when SSRS is installed or joins a farm. Figure 11-8 shows a section of the RSReportServer.config file,

which contains sensitive authentication credentials required to connect to SSRS server components. Notice that parts of the data inside the file are encrypted. SSRS uses the keys associated with the report server instance to decrypt the contents of this file as well as the encrypted content stored in the ReportServer database in the dbo.Keys table.

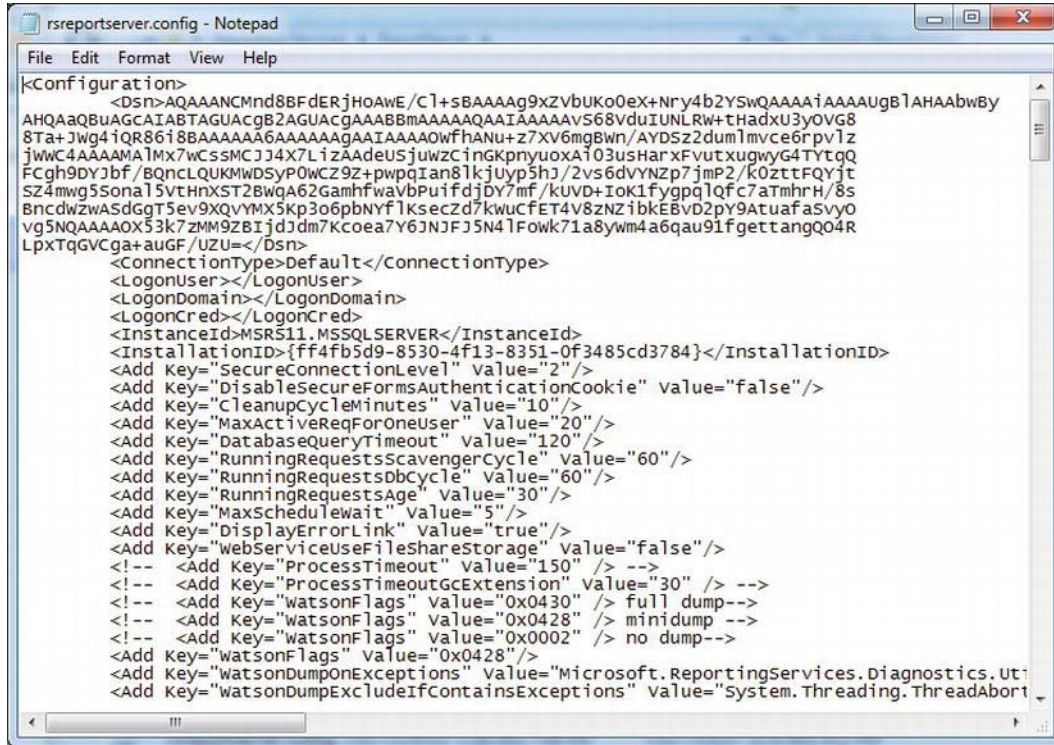


Figure 11-8. The rsreportserver.config encrypted values

We will show how to use both tools—the Reporting Services Configuration Manager and the RSKeyMgmt utility—to extract the key for the PROSSRS SSRS installation.

First, open the Reporting Services Configuration Manager again and connect to your report server instance (called MSSQLSERVER by default), and then click the Encryption Keys icon on the left. You will see four options on the Encryption Key page: Backup, Restore, Change, and Delete Encrypted Content. You can back up the encryption key to a key file and supply a password, as shown in Figure 11-9. This file should be stored in a secure location. If the report server ever had to be rebuilt for any reason—because of a hardware failure or data corruption issue—then having this key is crucial to restoring the ReportServer database to its previous state. Without the key file, it is still possible to restore and initialize the ReportServer database from backup. However, all objects that require encryption, such as data sources with stored account information, must be reset manually, which could be an arduous task at best.

To use the command-line tool that will essentially perform the same task of backing up, restoring, and deleting the encryption keys for the report server instance, the syntax is as follows:

```
RSKeyMgmt -e -f C:\Pro_SSRS\SSRS_Key\PROSSRS_SSRS_Key -P Password
```

The -e option tells RSKeyMgmt to extract the key to the file PROSSRS_SSRS_Key in the C:\Pro_SSRS\SSRS_Key folder. The password option is required and the password must meet the minimum complexity requirement. If you needed to do so, you could reapply the key to the server using the same command but changing the -e option to -a. After executing the command, you are timidly instructed to SECURE THE FILE IN A SAFE LOCATION!



Figure 11-9. Encryption keys in the Reporting Services Configuration Manager

The key will also come in handy in case you need to move the Reporting Services instance to another machine on your network. You can simply install SSRS on the new machine, restore the databases from the old instance, restore the key that you have been storing, and remove the original encryption key. This makes a simple standalone SSRS migration fairly simple and painless.

Setting Up Authentication and User Access to Data

Access to confidential electronic data, no matter where they reside, begins and ends with user authentication. Having security users or roles properly configured is critical to a secure deployment of SSRS. In an environment utilizing a Windows Server Domain, SSRS can then take advantage of the authentication provided by Active Directory's security groups and users. The SSRS administrator is responsible for configuring SSRS-specific security roles that link to Active Directory security accounts. In the following sections, we will show how to set up a test Windows account for an employee who will have limited access to the SSRS report server. We will discuss the following:

- *Setting up SSRS roles:* SSRS roles dictate what permissions the users will have when they access the SSRS server. An Active Directory security account, either a group or a user, is assigned either to one of five predefined SSRS roles or to a new role that the SSRS administrator may create.

- *Assigning SSRS roles:* Assignments are the actual SSRS tasks that a user in a specific SSRS role may perform.
- *Configuring and testing permissions for SSRS objects:* Each report folder and its objects maintain individual permissions that can be set at the folder level and propagated to all children objects or that can be set specifically per object. We will show how to set up two folders for the test user account and add report objects that are to be secured.
- *Filtering reports:* It is possible to limit which data are displayed within a report, based on the Active Directory login account that is accessing the report server. You do this by associating the value returned from an SSRS global collection, `User!UserID`, with a field value in the dataset of the report; `User!UserID` returns the current login account.
- *Authenticating data sources:* In addition to the Windows login account and SSRS role assignments, data sources maintain their own authentication properties, which we will discuss.
- *Setting permissions on the data source database objects:* You may recall from an earlier chapter that you created a stored procedure, `Emp_Svc_Cost`, to use with the Employee Service Cost report but did not assign user-specific permissions. We will show how to assign the permissions settings in this chapter.

Introducing SSRS Roles

By default, the installed SSRS Web service uses Windows integrated authentication to access reports and report content. Windows user or group security accounts stored in Active Directory must be associated with an SSRS role before they will have access to the SSRS server. Administrators can assign the Windows accounts to SSRS roles with Report Manager. In the test scenario for our health-care application, we have set up a test Windows account, named `jyoungblood`; you will assume `jyoungblood` is a registered nurse in a health-care organization who makes home visits to patients.

All the clinical staff, including nurses such as `jyoungblood`, are associated with security groups within Active Directory for the domain. So, you will make `jyoungblood` a member of the `RNsecurity` group. In addition to the security group `RN`, all registered nurses, including `jyoungblood`, will be contained within an organizational unit (OU) inside Active Directory, as shown in the Active Directory Users and Computers window in Figure 11-10. Although you will not use OUs when assigning a user or group to a role in SSRS, it is important to note that you can use OUs to configure Group Policy settings that apply to security as well, such as locking down the user's desktop or Internet Explorer.

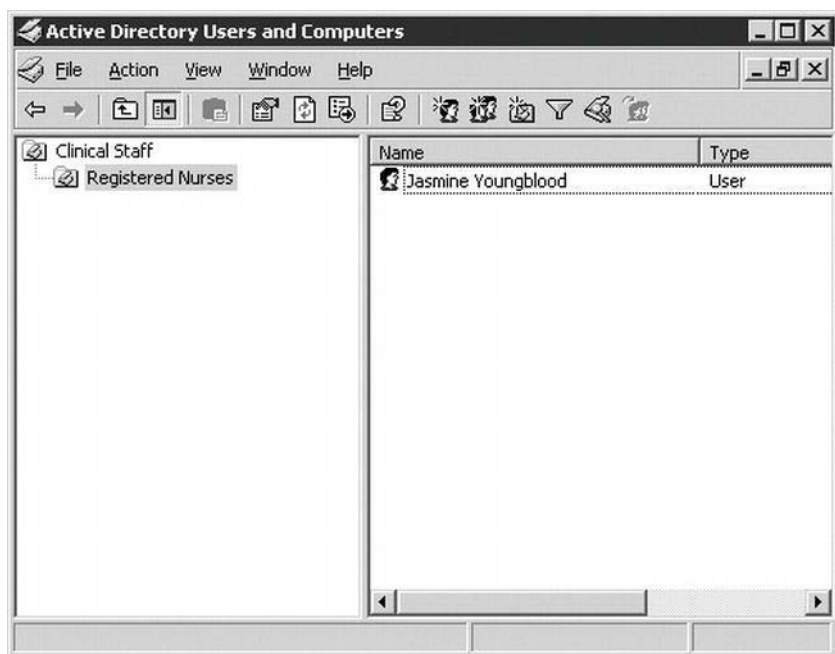


Figure 11-10. Test Windows account in Active Directory

Before assigning the test Windows user to an SSRS role and testing the permissions settings, first look at these five predefined roles:

- *Browser*: Users assigned to the Browser role may only view reports, folders, and resources. They may also manage their own subscriptions.
- *Content Manager*: Administrators are assigned to the Content Manager role by default. Users assigned to this role can perform every task available for SSRS objects, such as folders, reports, and data sources that they manage.
- *My Reports*: This is the default role automatically assigned to a user when the My Reports feature is enabled on the SSRS server, discussed later in this section.
- *Publisher*: Users assigned to this role, by default, have enough privileges to publish reports and data sources to the report server. Typically, this role is used for report authors who work with Report Builder or Visual Studio to create and deploy reports.
- *Report Builder*: The Report Builder role is used primarily for assigning the required permissions to users who will use the Report Builder application launched from Report Manager, which is covered in Chapter 09.

SSRS roles are defined by the tasks that users assigned to each role may perform. SSRS tasks provide content management permissions and define which SSRS objects are viewable by the user. Users can perform the following tasks:

- Consume reports
- Create linked reports
- Manage all subscriptions
- Manage data sources
- Manage folders
- Manage individual subscriptions
- Manage models
- Manage report history
- Manage reports
- Manage resources
- Set security for individual items
- View data sources
- View folders
- View models
- View reports
- View resources

Each predefined role is configured by default, with a specific combination of allowable tasks. Users assigned to the Publisher role, for example, may manage folders, reports, resources, models, and data sources as well as create linked reports.

SSRS 2012 does not allow administrators to edit the tasks available to roles or to add new roles through the Web-based Report Manager. That work is relegated to the SSMS 2012 interface. Open SSMS and connect to the Reporting Services instance by using the Connect button and selecting Reporting Services. The dialog box will look similar to the normal connection screen, but the server type will be Reporting Services. Enter the name of the SSRS server and click Connect to view the available options. Figure 11-11 shows the connection screen.

Once connected, you will find three sections listed for the instance. The section we will focus on here is the second, Security. Expand the Security node to reveal two subnodes, Roles and System Roles. The Roles section is used to manage user access on a reporting level. This section handles regular users who need to view, browse, and create reports. The second section, System Roles, is used to administer privileges for overall system tasks. We will focus on the Roles section here.



Figure 11-11. Connecting to SSRS via SSMS

Expand the Roles node to see the five default roles that we discussed earlier. To see the properties of the role, right-click the browser and select Properties. You will see a description for the role, along with each of the tasks that the role can have. Figure 11-12 shows you how this will look. From this window, you can edit the description or edit what tasks the role can perform. You do not want to change the Report Builder role tasks now, so have a look around to get a feel for what options you have with security changes, but click Cancel to discard any changes you might have made.

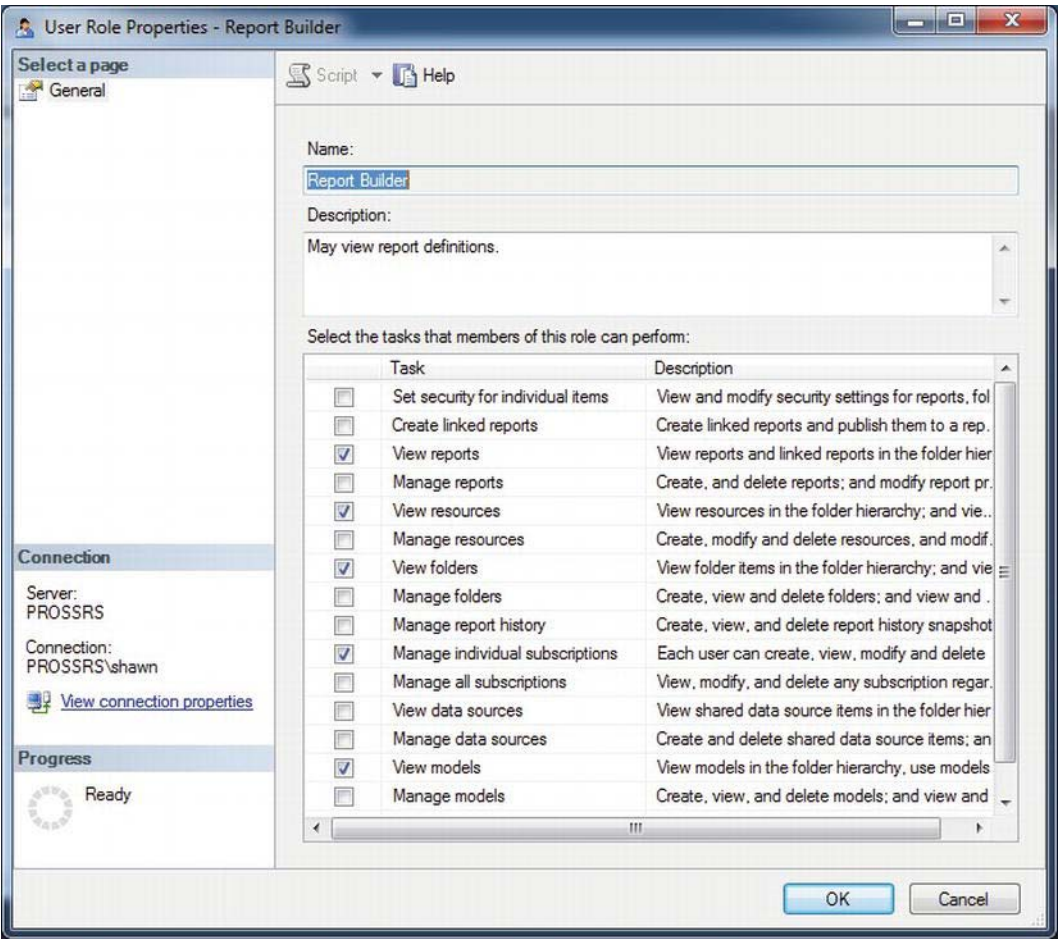


Figure 11-12. Viewing the Report Builder role task permissions

From SSRS, you can also add new roles or edit any existing roles for the server. This is important if you need to create a new custom role for users who may need permission that is slightly elevated from a standard SSRS role. This interface is identical to the role-editing window and can be reached by right-clicking the Roles node and selecting New Role.

Testing SSRS Role Assignments

In this section, we will go through the process of adding folders and report objects that would be in line with what nurse jyoungblood would use. You will want to ensure that she will not have the ability to navigate to other folders and run other reports that may contain confidential information.

The first step in testing jyoungblood’s access to the reports that have been defined for her security group, RN, is to publish the reports to a folder on the SSRS server that will contain reports for registered nurses. As the administrator for the test, open Report Manager, and create two new folders: one in the

root folder called Clinical Reports and then one inside the Clinical Reports folder called Registered Nurse. To do this, simply click New Folder in Report Manager. Because both of these folders, by default, are inheriting permissions from the parent folder, which currently is configured for administrator access only, you will alter the permissions manually so that the new folder (and the reports and data source you will add to it) will maintain its own security settings.

To publish reports to the Registered Nurse folder, you could use any method already covered in the previous deployment chapter, but for this test, simply upload a report you have already worked with: Daily Schedule. Then create a data source called RN_DS for the purposes of testing security. Upload the report file Daily Schedule.rdl from Report Manager in the Pro_SSRS project. In Report Manager, in the Registered Nurse folder, simply click Upload File, browse to Daily Activity.rdl, and click OK. To create the data source manually with Report Manager, simply click New Data Source in the Registered Nurse folder, name the data source RN_DS, and set the connection string to the following (replacing localhost, if necessary, with your SQL Server server name where you have created the Pro_SSRS database):

```
Data Source=localhost;Initial Catalog=Pro_SSRS
```

In this case, you will choose to set the data source authentication method to Credentials Stored Securely in the Report Server and supply a name and password that will be used to access the data in the Pro_SSRS database. Assuming that jyoungblood's Windows account was granted access to the data source database, you could have selected the Windows Integrated Security option to pass through the Windows account to the SQL Server database. You know that you will configure the report to filter out data that are relevant only to the clinician jyoungblood, so you don't need to be overly concerned with the stored credentials. If you don't already have a SQL Server login with permissions to the Pro_SSRS database, go ahead and create one called myreportuser and give it db_datareader permissions to the database. You can see the completed screen for setting up this new data source in Figure 11-13.

New Data Source - Report Manager

Name:

Description:

☐ Hide in tile view

☒ Enable this data source

Data source type:

Connection string:

Connect using:

☒ Credentials supplied by the user running the report

Display the following text to prompt user for a user name and password:

☐ Use as Windows credentials when connecting to the data source

☒ Credentials stored securely in the report server

User name:

Password:

☐ Use as Windows credentials when connecting to the data source

☐ Impersonate the authenticated user after a connection has been made to the data source

☐ Windows integrated security

☐ Credentials are not required

Figure 11-13. Setting up the RN_DS data source

Figure 11-14 shows the folder structure and setup of the report objects for the initial test you will perform. At this point, you have not granted SSRS role assignments to the Windows account, `jyoungblood`, or the security group, `RN`, of which she is a member. The Daily Activity report provides clinicians with a list of their daily activities. In a data-driven subscription, where the report can be mailed to the clinicians after processing, the parameter for the employee's ID was used to create reports with data unique to each individual. In this test now, however, you want to allow access to the same report to be run manually from Report Manager. This poses its own set of concerns, which we will cover as we step you through the process.

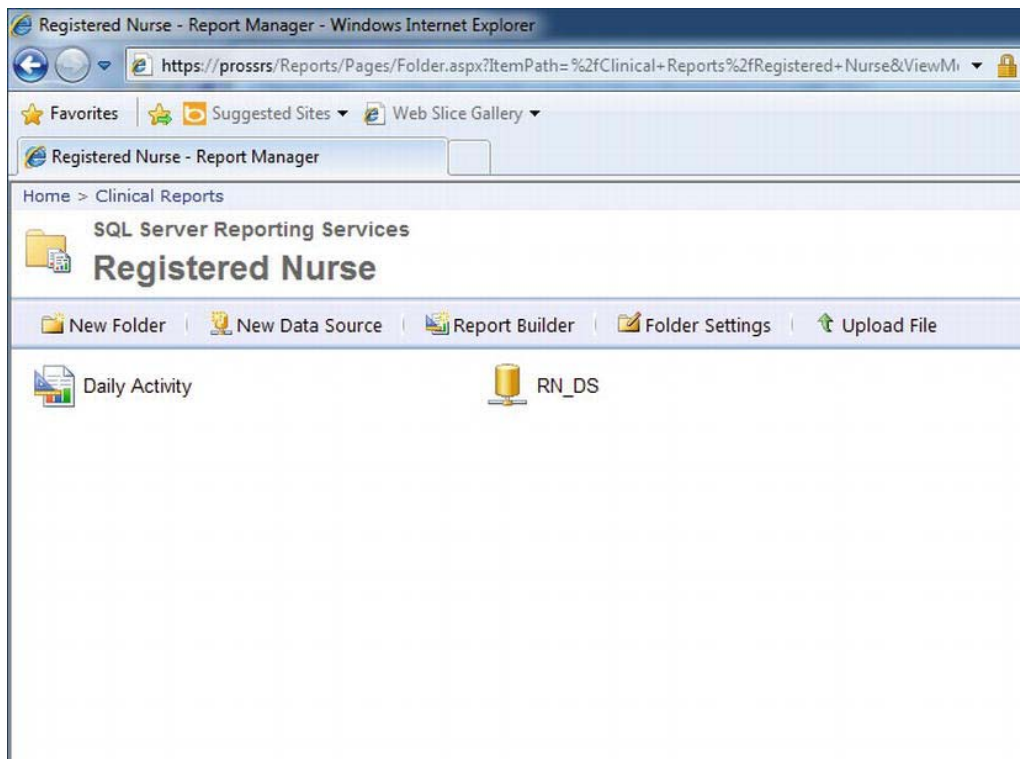


Figure 11-14. Report objects for registered nurse test

To begin the test, open a browser and log in with the user `jyoungblood`. Once the browser is open, paste the link in the address bar to the Registered Nurse folder you created previously, as shown here:

`http://YourServerName/Reports/Pages/Folder.aspx?ItemPath=%2fClinical+Reports%2fRegistered+Nurse&ViewMode=Detail`

You have to paste the link into the browser because the permissions for `jyoungblood` currently do not allow navigation to the report directly through Report Manager. As you can see in Figure 11-15, when you view this link, you receive an error message indicating that the user does not have permissions to view the resources in the folder.

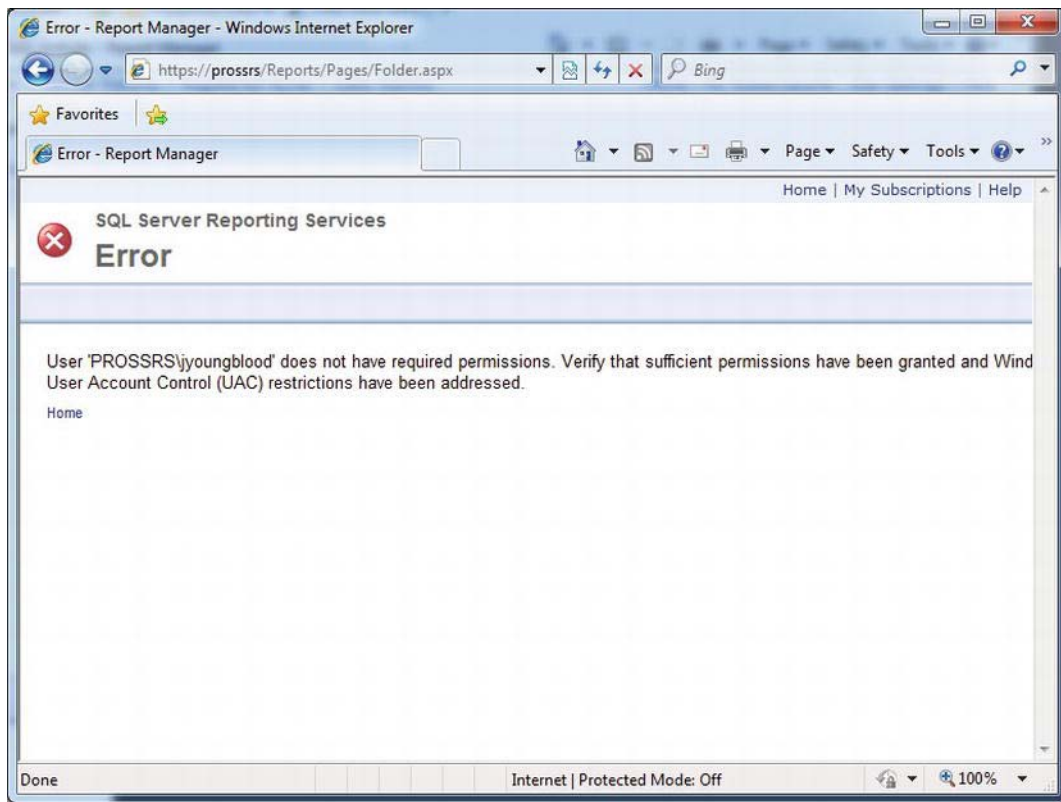


Figure 11-15. Error message for insufficient permissions

Running Report Manager as an administrator again, you are now going to set the permissions for `jyoungblood`. You can control security settings on the Property tab for each folder as well as on individual report items. In this case, you will set permissions at the folder level for the Registered Nurse folder. Navigate to the Registered Nurse folder while using a browser running from an SSRS Administrator account, click the Folder Settings button in the folder menu bar, and then select the Security option on the left. As you can see in Figure 11-16, the default security group is `BUILTIN\administrators`, which is assigned to the Content Manager role.

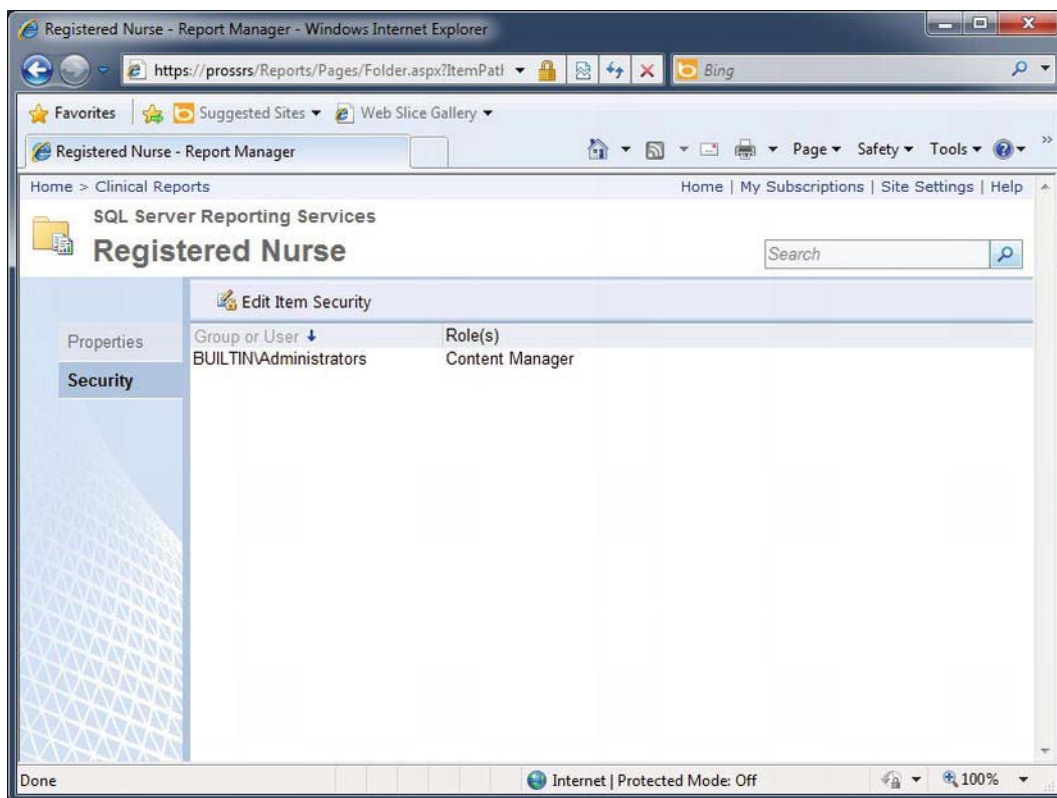


Figure 11-16. Registered Nurse folder's default permissions

To set permissions for the RN group, click Edit Item Security to break the security inheritance from the parent folder. Go ahead and click OK on the warning screen that you are breaking the security inheritance. Now you can click New Role Assignment and add the RN security group as the group name on the New Role Assignment form. For the role assignment, choose Browser, which will allow the users assigned to the RN group to view the Registered Nurse folder and all its child nodes, to view reports and resources, and to configure their own subscriptions. You can see this in the security list now in Figure 11-17.

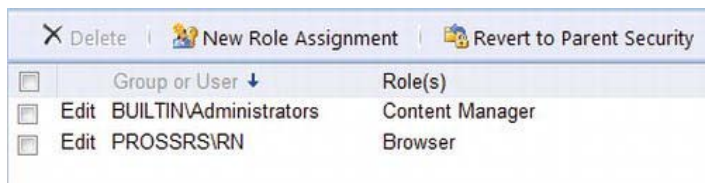


Figure 11-17. New security list for the Register Nurse folder

Now when you access the Registered Nurse folder logged in as jyoungblood, all you see are the reports that have been deployed to that folder, not the data source. In addition, all the properties for the objects you can view have limited accessibility and content. If you click the menu for that report item and select Manage, for example, you will see only the general properties information, such as the Creation Date and Modified Date of the Daily Schedule report. In contrast, an administrator viewing the same properties page would be able to see and modify other report property settings such as Parameters, Data Sources, Execution, History, and Security, as shown in Figure 11-18.

There are two SSRS roles that you should be familiar with. We have just mentioned one, System Administrator, and the other available SSRS role is System User. These are site-wide roles that allow users to perform certain tasks within the SSRS installation. These are not different from the folder/item level security we have discussed previously, in that these roles grant privileges to the entire SSRS system. You can assign these roles to users under the Site Settings section of Report Manager. The following is a list of what each role allows the user/group to do when assigned the role:

System User

- View system properties
- View shared schedules
- Allow use of Report Builder or other clients that execute RDL

System Administrator

- View/Modify system role assignments
- View/Modify system role definitions
- View/Modify system properties
- View/Modify shared schedules
- All other access granted to System User role

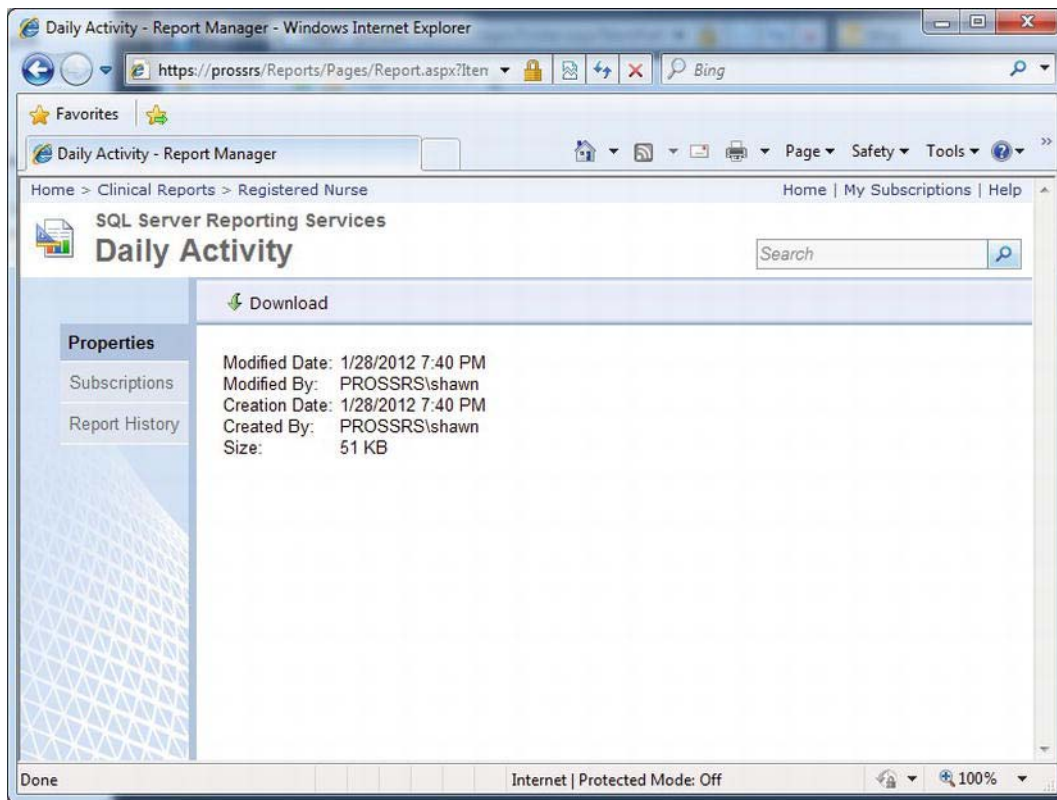


Figure 11-18. Report properties available for an administrator

To complete the test, you will simply execute the Daily Schedule report as `jyoungblood`. You have granted permission for the RN Windows security group to inherit the SSRS Browser role, so you should not have a problem executing the report. The report executes successfully. However, Figure 11-19 shows one glaring issue—even though `jyoungblood` has executed this report, she is seeing other employees' scheduled visits. Although she would be able to enter an `EmployeeID` parameter value that would limit the data on the report to only her data, she would still be able to see other employees' schedules by entering their IDs, assuming she knew what they were. Although this might be an acceptable practice for many companies, in the next section we will show how to go a step further to ensure that she will be able to view her schedule only.

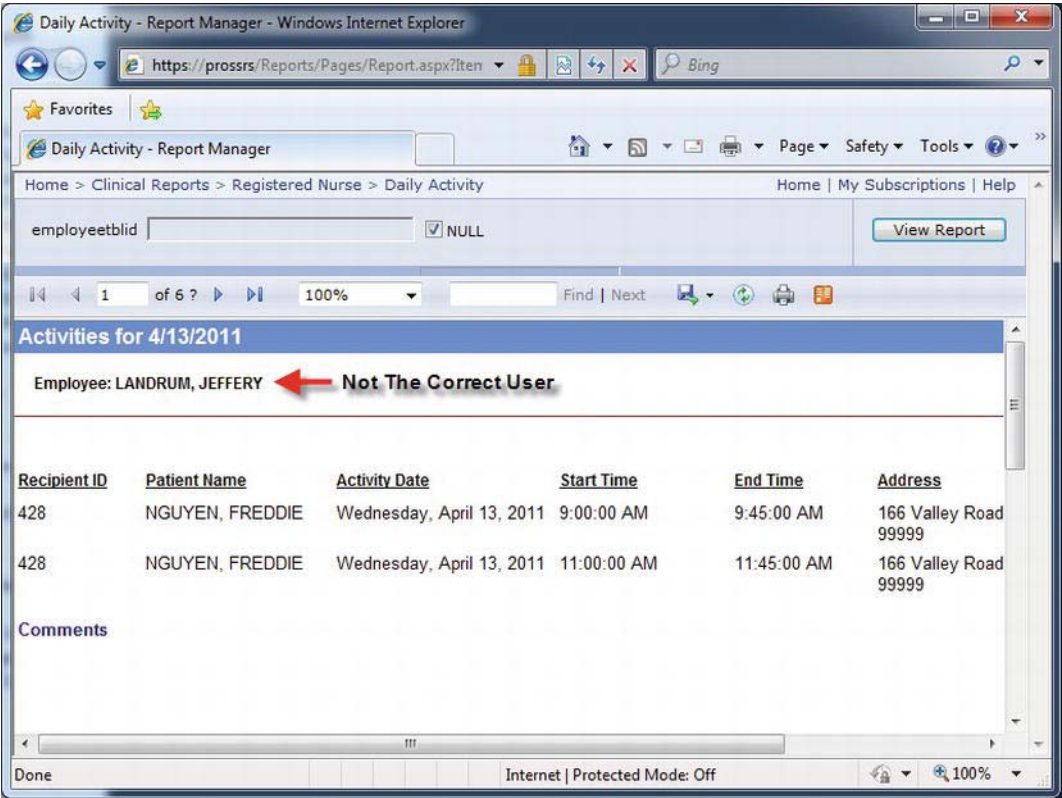


Figure 11-19. Test account viewing other employee's data

■ **Tip** When building a testing environment, a number of available resources can simplify the process. Virtualization is common in most every environment today. Microsoft's Hyper-V or other desktop virtualization tools are advantageous tools for testing, as they allow you to run multiple operating systems simultaneously on a single machine.

Filtering Report Content with User!UserID

For the Daily Schedule report, say you have decided that you want the users to be able to view only their own schedules. SSRS allows you to accomplish this by creating a report filter that uses the value of the login account for the user executing the report. The login name value is returned from a global collection in SSRS. You have been using global collections all along—for example, when you use an expression such as `=Fields!FieldName.Value`, you are actually returning a value from the Fields global collection.

The global collection that you will use for the report filter is User, and the value you are interested in is UserID. The expression will therefore be =User!UserID.

To use User!UserID in the filter, you will need a field in the dataset that will equal the UserID value. In the dataset for the Daily Schedule report, you may recall that you have a field called HWUserLogin that you can use for this purpose. When compared by the filter, the two values will be identical—one value delivered with the dataset and the other at execution time of the report. After the filter is applied, the report will display only those records where the username of the employee executing the report matches the value of the HWUserLogin field returned with each record of the dataset.

Unlike parameters, filters cannot be set through Report Manager. To set up a filter, you will need to modify the report itself, either in the RDL file directly, through Report Builder or in Visual Studio, as shown in Figure 11-20. Notice that you can use the RTRIM function to strip off the trailing spaces; otherwise, the comparison may fail. Make sure that the employee Login matches the user that you have created so that the comparison will match. In the book example, this is PROSSRS\Jyoungblood, but it may differ for your examples. Add the filter to the group properties to filter out any other users except the viewing user.

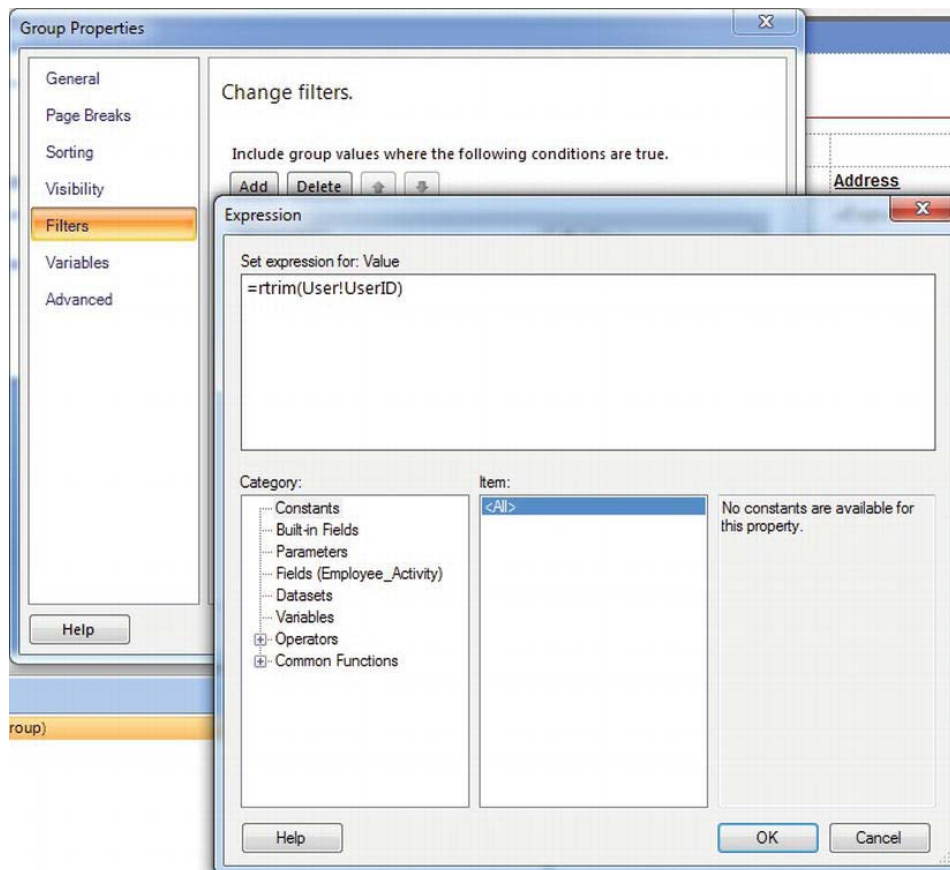


Figure 11-20. UserID filter for daily schedule report

Because this report may return several hundred records, even though it will filter automatically for each user, it is a good idea to cache the report for ten minutes. Caching will help alleviate the performance hit of requerying the data source every time a new user accesses the report. When the user jyoungblood executes the report again, you can see that the schedule now reflects only her schedule, as shown in Figure 11-21.

Recipient ID	Patient Name	Activity Date	Start Time	End Time	Address
473	RICHARDS, JORDAN	Saturday, December 11, 2010	7:55:00 AM	8:30:00 AM	104 Winding Wa 03865
68	ERICKSON, RENE	Saturday, December 11, 2010	7:45:00 AM	8:35:00 AM	418 Brianwood D MA 02368
391	WANG, DWIGHT	Saturday, December 11, 2010	8:45:00 AM	8:50:00 AM	382 Locust Lane NE 68601
399	MANNING, LEE	Saturday, December 11, 2010	8:00:00 AM	8:55:00 AM	118 Park Place RI 02895
13	HUBBARD, EDGAR	Saturday, December 11, 2010	8:05:00 AM	9:00:00 AM	14 Maple Street Hills, CA 95814
29	CHASE, DERRICK	Saturday, December 11, 2010	8:00:00 AM	9:00:00 AM	268 Valley View Fe, NM 88044

Figure 11-21. Filtered report execution for a test user

Setting Data Source Security

Once you have deployed the data source to the report server, you can specify its connection properties. This is an important step because the property settings determine how both the user and SSRS will connect to the data source. When executing unattended reports—for example, for a user subscription—SSRS will control passing authentication credentials to the data source and must have access to valid authentication credentials.

Four connection options for the data source are available in Report Manager:

- *The Credentials Supplied by the User Running the Report:* With this option, users are always prompted to log in to the data source when executing the report.

- *Credentials Stored Securely in the Report Server:* SSRS uses authentication credentials stored in the ReportServer database. The sensitive login information is encrypted.
- *Windows Integrated Security:* This option passes the login information for the current user to the data source. Don't choose this option if the data source will be used for unattended installs or if Kerberos is not configured for the Windows domain. You will also get issues when using this where multiple hops are made from SQL Servers. Unless Kerberos is installed and configured correctly across your entire domain, this may not work correctly in all situations
- *Credentials Are Not Required:* This is the least secure option and is used when the data source does not require authentication.

Setting SQL Server Permissions

In Chapter 3, when you created the stored procedure called `Emp_Svc_Cost`, you set the permissions to allow public execution while designing the report. The environment you were working in was otherwise secure, as it was isolated from other networks and there was no fear of it being compromised.

Now that you are deploying the stored procedure in a production environment, you will need to lock down the stored procedure as well. You can do this through SSMS by right-clicking the stored procedure and selecting Properties (see Figure 11-22). Next, click the Permissions page. Search for the RN group using the Search... button to load its explicit permissions. Check the Execute permission under the Grant column and this will allow any user from the RN group to execute this stored procedure. You do not need to explicitly grant Execute rights to the test user `jyoungblood`, as she is a member of the RN security group.

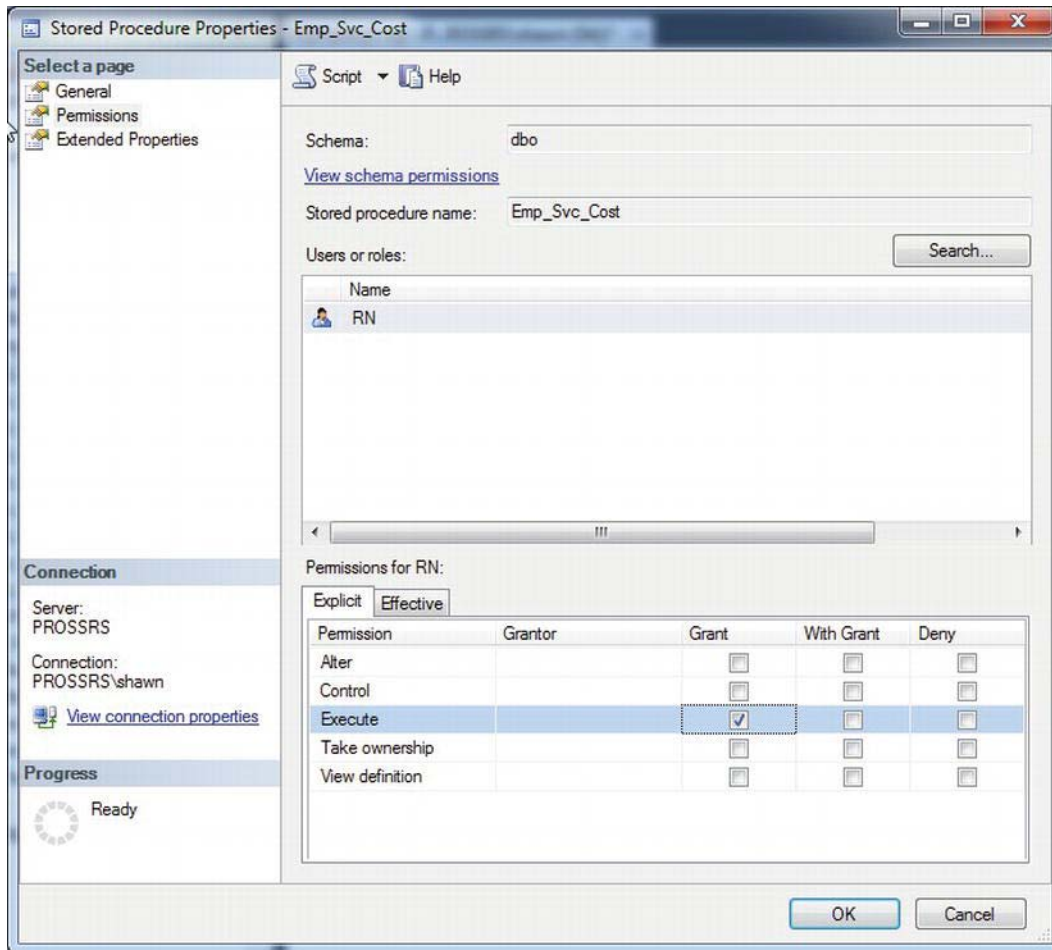


Figure 11-22. Setting SQL Server stored procedure permissions

Auditing Reports

Having the ability to know the details of report execution, specifically for undesired access, is an essential piece of the security puzzle. In this chapter, we are going to use the built-in report execution logging and a report management table to build a report that will show up who ran what reports and much more information. Having an audit trail of report execution is essential for gaining insight into user behavior and possible security breaches.

SSRS Auditing

In this section, we'll show how to use the built in views in the ReportServer database to audit the following in SSRS:

- *Report execution activity:* Which user executed which report and when, and what status the report finished with (to find possible issues)
- *Parameter inputs for reports:* Which parameters were entered by a user

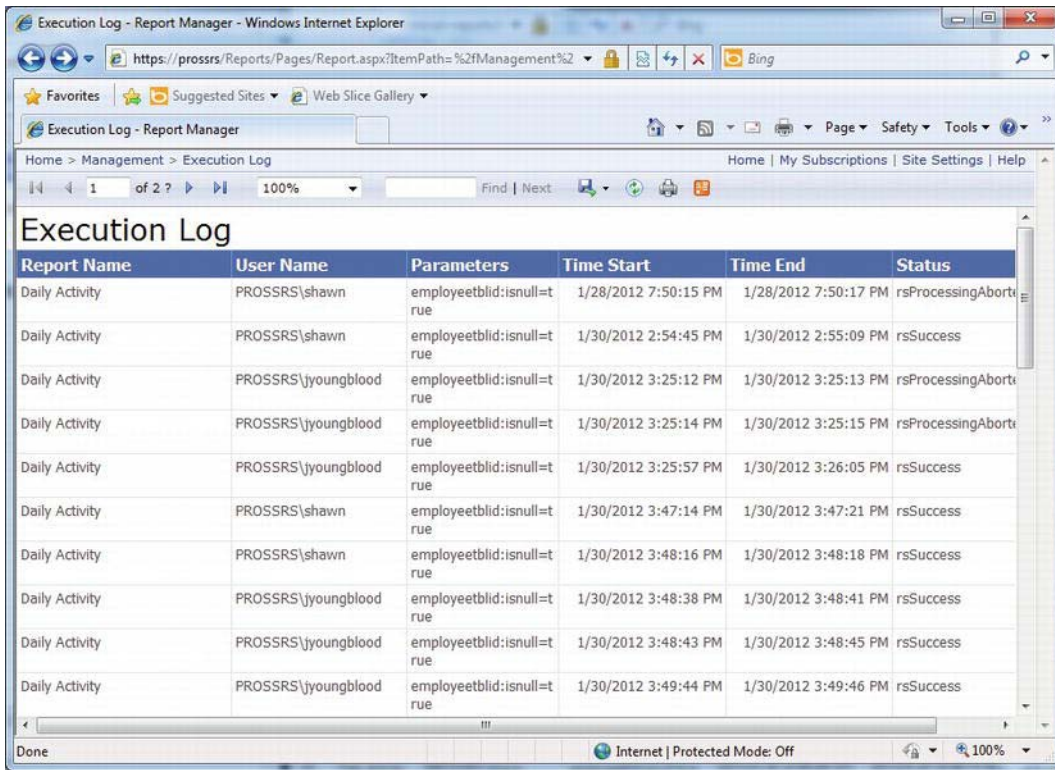
The latter is important because even though SSRS contains validation for certain types of parameters, such as Integer and DateTime, when a report parameter uses a string, it is susceptible to a SQL injection attack. SQL injection attacks are made possible when a Web page or service takes input from a user or program that could contain injected code that could execute on the SQL Server machine. These types of attacks can be malicious in nature or can cause the report or page to deliver more data than was intended.

We have simply made the report a table instead of a matrix, which was more in line with the PivotTable analysis for performance. In this modified report, called Report Execution Audit (which is available in the Pro_SSRS chapter 11 files), you will also include the parameters that the user has selected. Go ahead and upload this report and a new datasource pointing to your ReportServer database in a new folder in the root of your SSRS installation called Management. You will also need to point the report to this new datasource that you are creating. You can see the query we are using to pull this information in Listing 11-1.

Listing 11-1. Execution Log Query

```
SELECT
    Catalog.name
    , ExecutionLog.UserName
    , ExecutionLog.Parameters
    , ExecutionLog.TimeStart
    , ExecutionLog.TimeEnd
    , ExecutionLog.Status
FROM ExecutionLog
INNER JOIN Catalog ON ExecutionLog.ReportID = Catalog.ItemID
```

In Figure 11-23, you can see the times that the selected reports were executed as well as the parameter name and value that were entered. In addition to showing how the user interacted with the report via the Parameter field, the Report Execution Audit report also shows when the report was started and ended. Having this level of auditing is a valuable ally in the struggle to maintain security for confidential information. With HIPAA, it is also necessary to maintain an audit trail of user access to data. If you are suspicious that someone is accessing information they are not authorized to view, this report can serve as the audit trail, along with other normal auditing procedures such as the Windows event log.



The screenshot shows a web browser window titled "Execution Log - Report Manager - Windows Internet Explorer". The address bar shows the URL <https://prossrs/Reports/Pages/Report.aspx?ItemPath=%2fManagement%2f>. The page content displays the "Execution Log" with a table of report execution details. The table has columns for Report Name, User Name, Parameters, Time Start, Time End, and Status. The data shows multiple "Daily Activity" reports executed by users "PROSSRS\shawn" and "PROSSRS\jyoungblood" on 1/28/2012 and 1/30/2012. The status of the reports varies, including "rsProcessingAborted" and "rsSuccess".

Report Name	User Name	Parameters	Time Start	Time End	Status
Daily Activity	PROSSRS\shawn	employeeetblid:isnull=true	1/28/2012 7:50:15 PM	1/28/2012 7:50:17 PM	rsProcessingAborted
Daily Activity	PROSSRS\shawn	employeeetblid:isnull=true	1/30/2012 2:54:45 PM	1/30/2012 2:55:09 PM	rsSuccess
Daily Activity	PROSSRS\jyoungblood	employeeetblid:isnull=true	1/30/2012 3:25:12 PM	1/30/2012 3:25:13 PM	rsProcessingAborted
Daily Activity	PROSSRS\jyoungblood	employeeetblid:isnull=true	1/30/2012 3:25:14 PM	1/30/2012 3:25:15 PM	rsProcessingAborted
Daily Activity	PROSSRS\jyoungblood	employeeetblid:isnull=true	1/30/2012 3:25:57 PM	1/30/2012 3:26:05 PM	rsSuccess
Daily Activity	PROSSRS\shawn	employeeetblid:isnull=true	1/30/2012 3:47:14 PM	1/30/2012 3:47:21 PM	rsSuccess
Daily Activity	PROSSRS\shawn	employeeetblid:isnull=true	1/30/2012 3:48:16 PM	1/30/2012 3:48:18 PM	rsSuccess
Daily Activity	PROSSRS\jyoungblood	employeeetblid:isnull=true	1/30/2012 3:48:38 PM	1/30/2012 3:48:41 PM	rsSuccess
Daily Activity	PROSSRS\jyoungblood	employeeetblid:isnull=true	1/30/2012 3:48:43 PM	1/30/2012 3:48:45 PM	rsSuccess
Daily Activity	PROSSRS\jyoungblood	employeeetblid:isnull=true	1/30/2012 3:49:44 PM	1/30/2012 3:49:46 PM	rsSuccess

Figure 11-23. Report Execution Audit report

Introducing Log File Auditing

In addition to report auditing, SSRS supports two other types of logging: standard Windows event logging and trace file logging. You can use both of these resources to search for errors and warnings, as well as for other important information such as security information.

All of the trace files are stored in the default installation location `Drive:\Program Files\Microsoft SQL Server\MSRS11.MSSQLSERVER\Reporting Services\LogFiles`. Several types of log files exist: `ReportServerService` and `ReportServer`. Each file is named with a timestamp, such as `ReportServerService_05_31_2012_17_00_30.log` and `ReportServerService_HTTP_05_31_2012_17_00_30.log`, and contains information specific to its individual service. You can gather five levels of trace log information—0 through 4—that are controlled in the `ReportingServicesService.exe.config` file in the `Reporting Services\ReportServer\Bin` folder. Selecting 0 will disable tracing, and selecting 4 will enable verbose mode. Whenever an issue arises with SSRS, the administrator can generally isolate the problem by looking in either the event log or one of the trace files.

Summary

In this chapter, you examined three of the security tasks essential to secure SSRS deployment: encrypting, authenticating, and auditing. Internally, only employees who need access to confidential data should have it, and you can ensure this by applying a special filter for a report deployment. As a company that works heavily with confidential patient information, we are required to conform to the regulations imposed by HIPAA. However, in other industries, many similar regulations exist. Having security policies and procedures in place for any company is good practice, even when not working under stringent regulations. Fortunately, SSRS is designed to use the core-level security mechanisms that already exist in your organization through Windows authentication and can be extended, when required, to support other means of custom authentication.

Delivering Business Intelligence with SSRS

Most companies accumulate business data that, if analyzed correctly, can provide insights into what direction the company should take to achieve ultimate success. The main intention of Business Intelligence (BI) is to provide data in such a way that it can be immediately utilized to make important decisions. Microsoft's BI platform comprises many services and applications that work together to facilitate the analysis and delivery of critical business data. SQL Server is at the heart of the BI model, providing data storage, data transformation, notification, scheduling, analysis, and reporting services.

Having the right data available is just the first part of the challenge in building an effective BI system. The second part is to ensure that this data is delivered in an effective and accessible way to all the people who need to see it in order to make the right decisions. This is where SSRS comes into play. In our organizations, we found that by integrating SSRS with many of the other components of the BI platform, we were able to dramatically improve our overall business strategies by making necessary information available to our employees wherever they were and whenever they needed it.

In this chapter, you will examine the following applications that we have extended to include the SSRS reports for our software development company:

Analysis Services: Having the ability to “slice” through dimensions of data often renders unexpected and meaningful results. When OLAP Services was introduced with SQL Server 7.0, we were asked to build a data warehouse, transforming our OLTP data into an OLAP cube. We maintained this project using the version of Analysis Services that shipped with SQL Server 2000 and had some new functionality, such as data mining models. Microsoft has invested heavily in OLAP technologies ever since. One such enhancement is the VertiPaq engine that is used with SharePoint 2010, PowerPivot, and Excel Services. Using PowerPivot models and deploying them to a configured SharePoint 2010 server, an OLAP cube is spun up in memory to allow rapid slicing and dicing of hundreds of millions of records.

SharePoint Services: Part of Microsoft Office, SharePoint Services provides our company with an intranet portal that we have departmentalized. Any information relevant to individual departments or to the company as a whole is indexed and searchable. Integrating SSRS reports with SharePoint lets our employees easily find the data they require to do their jobs. I will show you how to add SSRS reports to SharePoint. Since SSRS builds on Windows SharePoint Services, the work you will do in this chapter will also be applicable to the Windows SharePoint Services included with Windows Server 2003. For a SharePoint 2010 and Reporting Services 2012 installation, you will need a 64-bit version of Windows Server 2008 SP2.

Most companies have similar applications to the ones we are describing. We are providing these examples to give you some ideas of how, with a modicum of effort, SSRS can easily enhance these types of business applications. Our purpose is not necessarily to provide a step-by-step guide, but to show how your company might use SSRS. If you use any of the technologies mentioned in this chapter, you can easily integrate this chapter's ideas into your own environment.

Building SSRS Reports for SQL Analysis Services

When we began the journey of building an OLAP solution for our health-care application with SQL Server 2000 Analysis Services, we were eager to jump right in and start analyzing data the first morning. We quickly became the resident experts at developing SQL queries to interrogate our OLTP database. Since this was the source database from which we were going to build the warehouse, our team thought it would be a simple case of adding a few queries and processing the cube. It did not turn out to be quite that easy. In retrospect, however, the process of creating an OLAP cube from a known source of data was worthwhile, because we were able to apply the skills we learned to many other projects. With each new version of Analysis Services, more and more features were delivered. At first, this can seem overwhelming, simply because of the volume of enhancements and the time required to not only become familiar with the technology, but also master it. However, while adding new features, Microsoft also adds many new tools that simplify time-consuming tasks. In the case of Analysis Services 2005, these new tools included a graphical MDX query builder and a cube wizard that automates many of the steps that create the intricate parts of an Analysis Services solution. SQL Server Analysis Services 2008 and 2008 R2 went even further to add time-saving functionality, enhanced data analysis, and the capability to integrate with SharePoint. SQL Server 2012 Analysis Services brought on new additions like the VertiPaq Engine mentioned earlier and Tabular Models, which are based off of the new Business Intelligence Semantic Model (BISM).

In this section, we will show how to use a simple Analysis Services cube as the data source to build and deploy SSRS BI reports. The cube is based on a SQL Server database that serves as a data warehouse for the health-care application you have been using throughout the book. The cube is populated with data relevant to patient admissions for a health-care agency. Though we designed the report to analyze many aspects of patient admission history, such as patients with multiple recurrent admissions, changing diagnosis, and patient referral sources, we will show how to create a report that specifically delivers analytical information about the length of time between when a patient is referred to the agency and when he or she is actually admitted. Over time, the data that is collected can help assist decision makers isolate problem areas and improve the processes that may be causing inefficient patient referral times. First, let's look at the database and cubes on which the report you create will be based.

The data warehouse database you will use as a source for the Analysis Services project is called `HW_Analysis`. It is a simple database containing only eight tables, and is populated with data using SQL Server Integration Services (SSIS). The typical process for preparing a data warehouse database with SSIS is to export data from the source OLTP database, transform the data to make it more conducive to analysis by SSAS, and finally load this transformed data into the data warehouse or data mart.

We have already built both the `HW_Analysis` database and the SSAS cube called Patient Referral, and we have included the required files for deploying these two key components and detailed installation instructions in the `ReadMe.txt` file included with code download for this book. The source code and related files are available in the Source Code/Download area for the book on the Apress Web site (<http://www.apress.com>). Once you have restored the `HW_Analysis` data warehouse database per the instructions in the `ReadMe.txt` file, you can open the `Pro_SSRS` project, which contains the Patient Referral cube. Figure 12-1 shows the simple Patient Referral cube structure in BIDS. The cube has six dimensions and two measures.

■ **Note** If you have skipped to this chapter, note that Business Intelligence Development Studio (BIDS) and SQL Server Data Tools (SSDT) have been used synonymously throughout this book. If you are using SSRS 2005 through SSRS 2008 R2, the shortcut to start DEVENV.exe was labeled SQL Server Business Intelligence Development Studio. In 2012, it is now labeled SQL Server Data Tools.

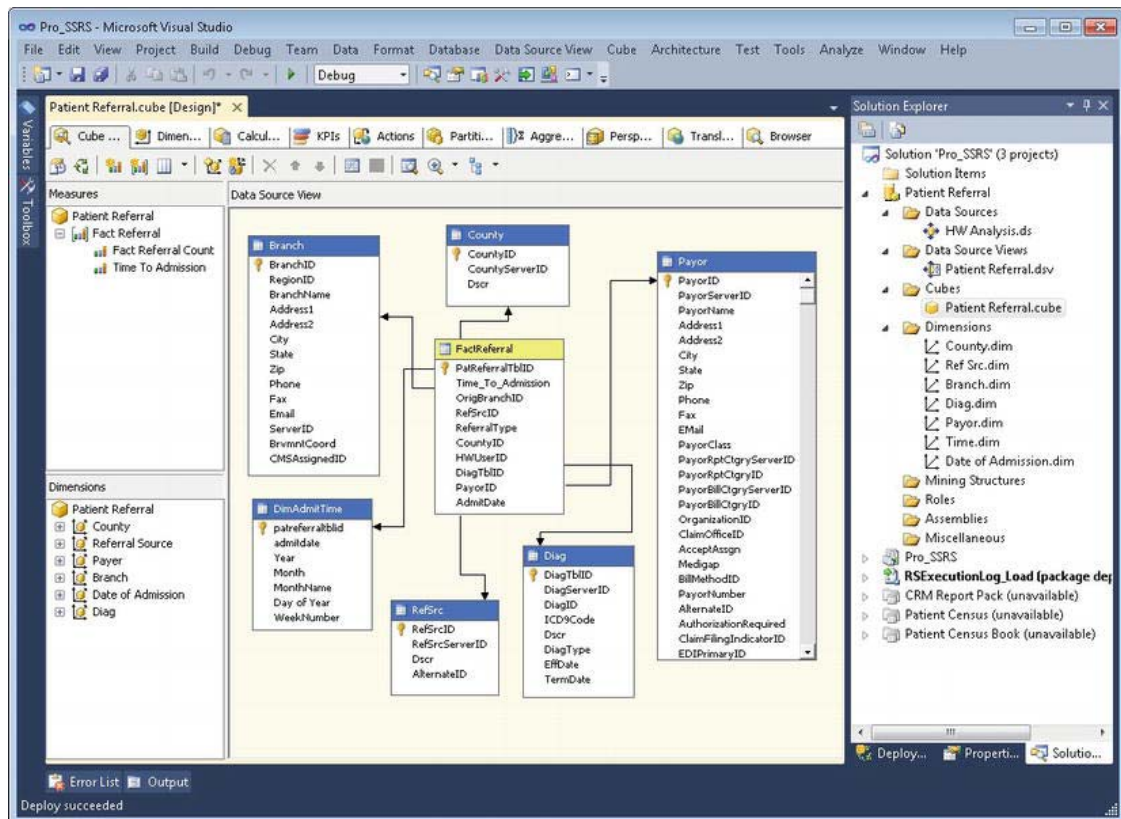


Figure 12-1. Patient Referral cube structure

Of the six dimensions, several of them should be familiar to you because you have worked with them in other chapters of the book. Dimensions contain textual attributes that describe the fact data. You can think of a dimension in a cube as pre-aggregated groups or categories of data that can be associated with one or more measures. Measures, such as Time to Admission and Fact Referral Count, are calculated values at each level in a dimension. Multidimensional data stored in a cube consists of both dimensions and measures that, when queried using MultiDimensional eXpressions (MDX), can be

analyzed not just two-dimensionally like rows and columns of a T-SQL query, but also by being drilled into and sliced at many dimensions.

Picture a cube like the Rubik's cube invented in 1974 that many of us played with as children. It contains three squares in height, three squares in width and three squares on depth. With that picture in your head, imagine if you will that the green side of the Rubik's cube represents the Date of Admission dimension, the red side representing the Branch dimension and the yellow representing the Time to Admission measure. Much easier than solving a Rubik's cube, you could create an MDX query that would return the intersection of these three points as shown in Figure 12-4.

The dimensions you will use in the Patient Referral cube are:

- County
- Referral Source
- Diagnosis
- Payer
- Branch
- Date of Admission

Since the starting point in this exercise is a prepopulated and processed cube, it is important to make sure you are getting the desired data from the cube before creating your SSRS reports. With the Pro_SSRS project open in BIDS, navigate to the Patient Referral SSAS project in the Solution Explorer, expand to Cubes, and double-click Patient Referral.cube. This will open the cube in design mode. Once the cube is open, select the Browser tab. Figure 12-2 shows the opened cube with the combined measures and dimensions ready to be dragged and dropped into the query pane. If no data appears in the Browser tab, and a message indicates that there is a permission problem or the database does not exist, this usually indicates that the cube needs to be processed or the SSAS Patient Referral database needs to be deployed. To deploy the Patient Referral OLAP database (the cube) in BIDS, right-click the Patient Referral project, and select Deploy. If the default permission settings are insufficient for your Analysis Services instance, you may have to open up the Data Source Designer for the HW Analysis data source and change the Impersonation Information to use a specific user name and password. For instance, my settings are set to use "SQLBIGEEK\bmcDonald" for the Windows user with appropriate permissions to deploy the cube. Once you have deployed the Patient Referral OLAP database successfully, you can return to the Browser tab, and click the available link to reconnect to the database.

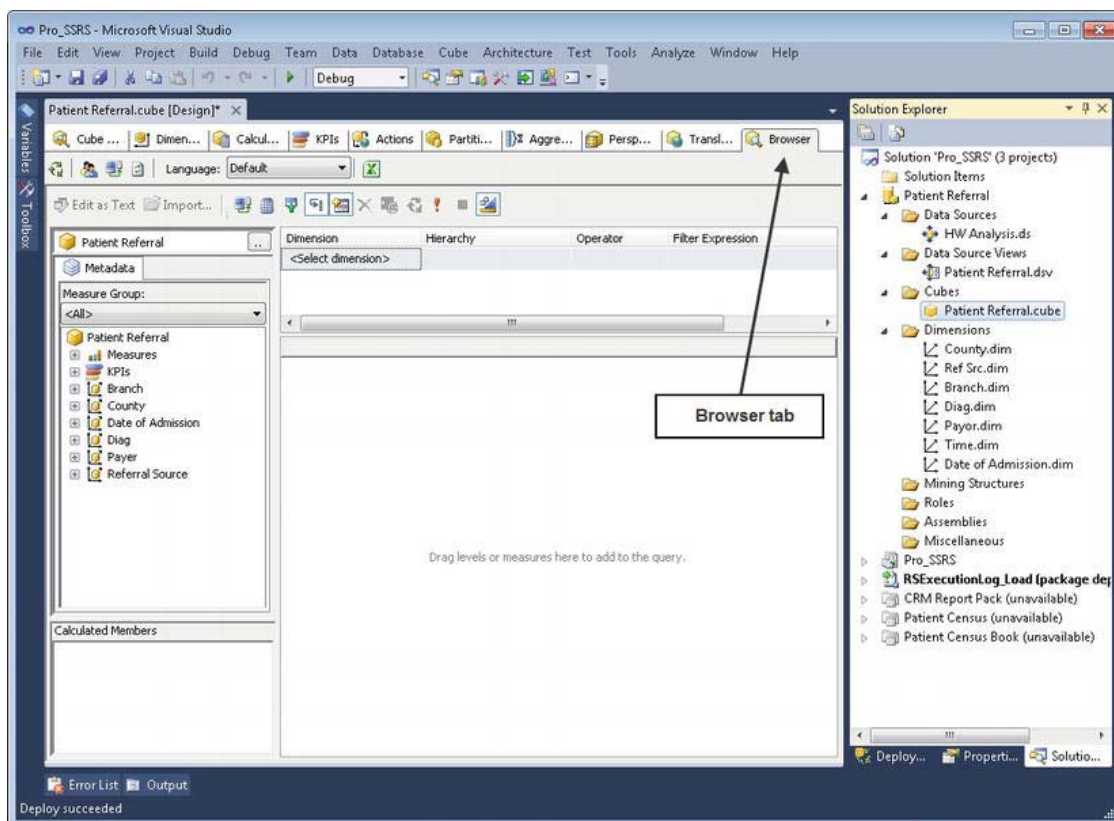


Figure 12-2. Viewing the Patient Referral cube in BIDS

Perform the following steps to see the average Time to Admission:

1. You can begin by expanding Measures and then Fact Referral in the cube object window on the left of the cube design pane.
2. Next, drag the Time of Admission measure to the Drag Levels or Measures here to add to the query pane. Typically, when browsing the cube or creating reports, you will want to drag over the measures before you drag over your dimension attributes. You should see an average time of 14.095 days. This calculation was done at the time the cube was originally loaded and processed and represents a total average of time to admission for all dimensions combined. Figure 12-3 shows the Time to Admission measure in the query pane.

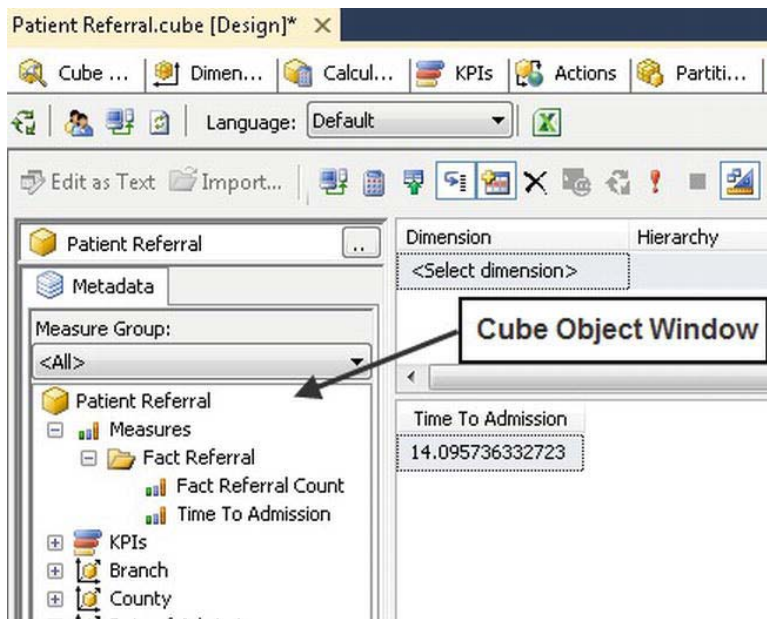


Figure 12-3. Viewing the overall Time to Admission measure's average for all dimensions

3. Now drag two dimension objects onto the data pane. Under the Branch object, drag and drop the Branch Name field to the left of our Time to Admission measure.
4. Next, under Date of Admission, drag and drop the Year - MonthName - admitdate hierarchy to the right of Branch Name in the data pane. The hierarchy looks like a pyramid made of six squares. When finished, the cube browser should look like Figure 12-4. You can now see the results for every single admitDate along with its corresponding Branch Name, Year and MonthName attributes.

In prior releases of the cube browser, you could drill down into hierarchies and have column groupings as we have seen in the Matrix-style reports. However, as of the current release of SQL Server 2012, the cube browser returns the data two-dimensionally much like the results of a T-SQL query. Some active members in the SQL Server user community has stated that this is a strange setback for Microsoft's cube browser and I personally believe that they should add the functionality back. However, nothing really has changed from the reporting perspective. If you are seeing the results depicted in Figure 12-3 above, then you are indeed getting data from the cube! Now it is time to move on to SSRS and create the report you will deploy to SharePoint.

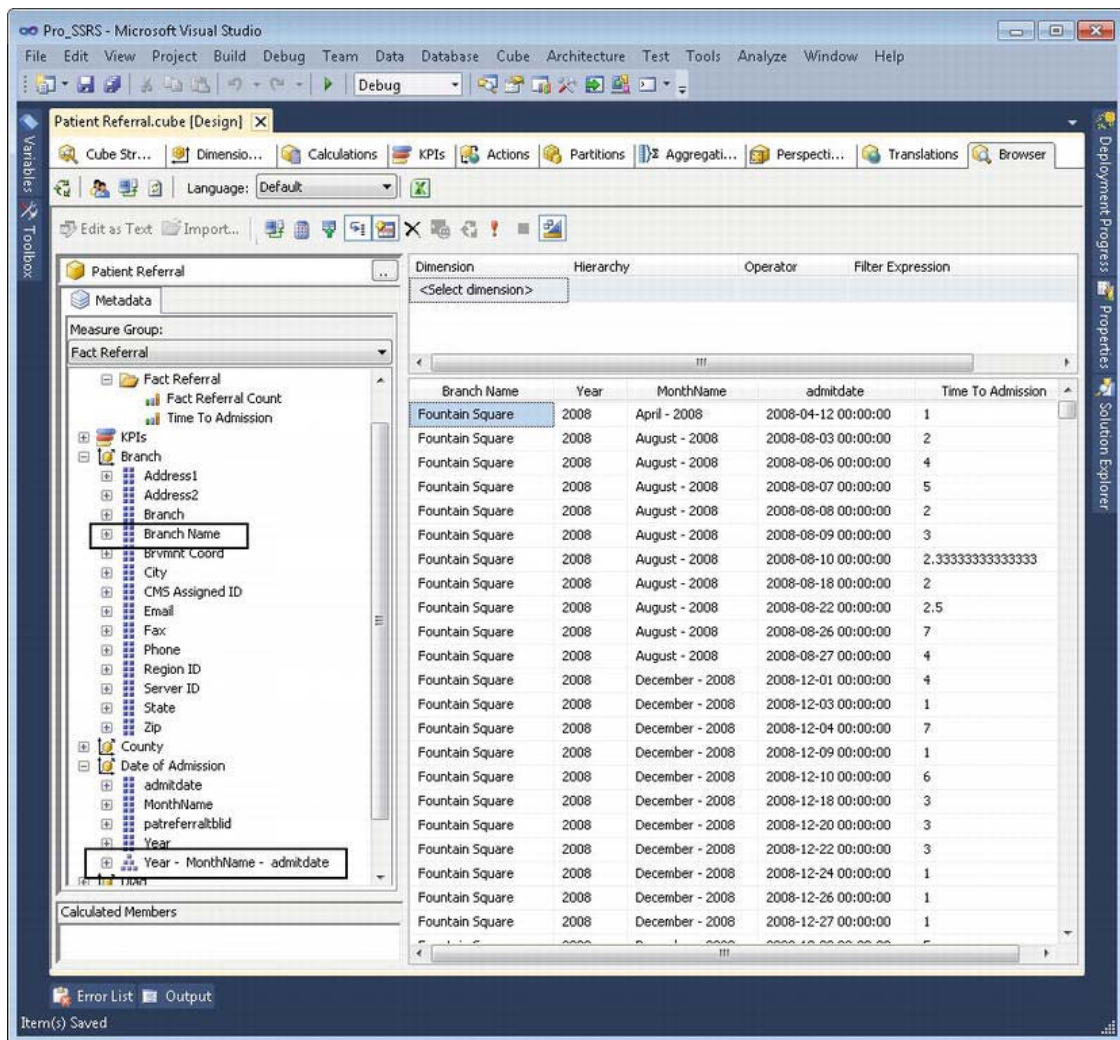


Figure 12-4. Cube browser with year and branch

Using an Analysis Service Cube with SSRS

Now that you have determined that the Patient Referral cube is working as it should, the next step is to create an SSRS report. In the Pro_SSRS project, you will find two completed reports, the Average Referral to Admission report and the Referral to Admission Chart report. Both reports are based on the Patient Referral cube. Since creating reports is the same, we will not go through the procedure of building these reports step by step. However, we will demonstrate the following through the Average Referral to Admission report:

- Setting up a data source to use Analysis Services

- Using the graphical MDX query builder that was added with SQL Server Reporting Services 2005

Setting Up the Analysis Services Data Source

With the Pro_SSRS project open, navigate to the Reports folder, and open the Average Referral to Admission report. Next, open the Report Data window. Right-click the Patient_Referral dataset and select the Query option from the submenu list. Notice that the query recognizes that you are using an OLAP data source with MDX and brings up the graphical MDX query designer. As you can see, it is very different from the T-SQL query window that we have used when accessing queries throughout the book. If you select filter expression and choose to filter on 2010, 2011, and 2012, you should see results similar to those shown in Figure 12-5. There are two datasets contained in this report, Patient_Referral and DateofAdmissionYear.

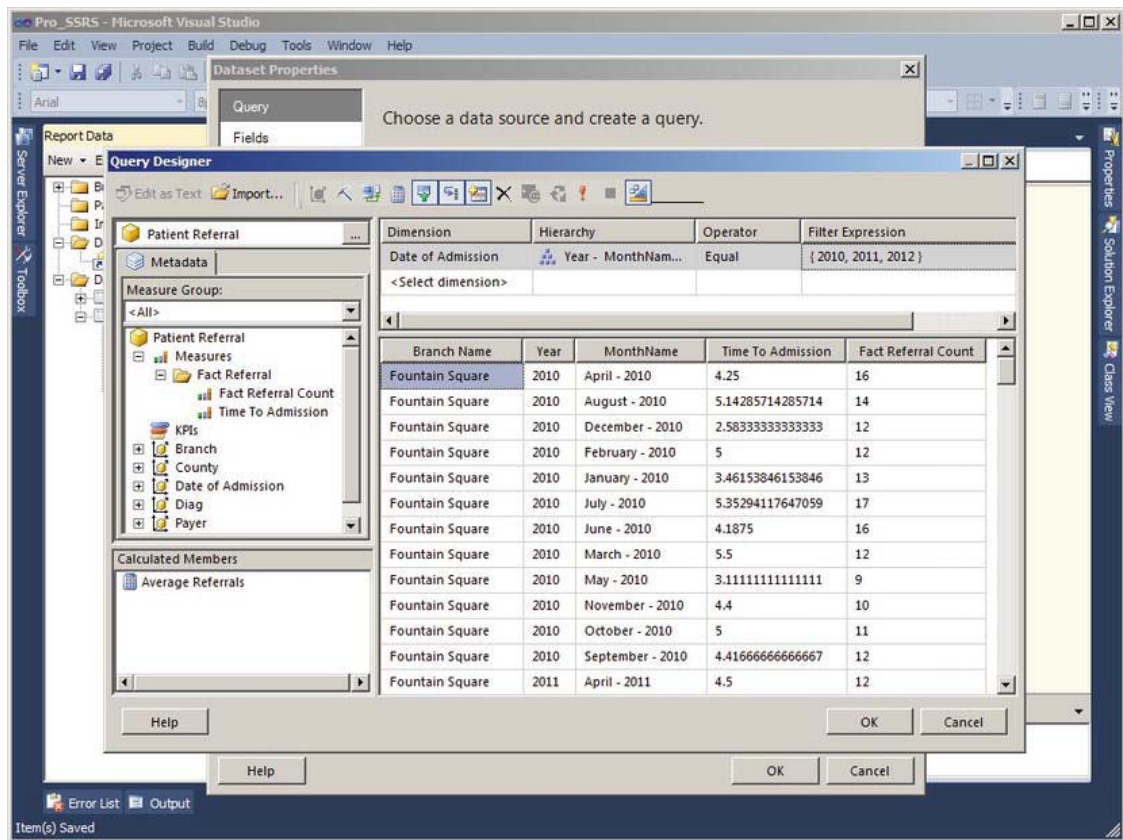


Figure 12-5. Viewing the Patient_Referral for the Average Referrals to Admission report

The type of data source for both datasets is Microsoft SQL Server Analysis Services. If your Analysis Services instance is not on the local machine that you are connected to, you will need to change the data source server from localhost, defaulted in the report, or the default to your SSAS server name. You can

do that easily by double-clicking the Shared Data Source named PatRef_DS and editing the data source properties to match your environment, as shown in Figure 12-6.

With the datasets configured properly for your SSAS environment, you will next look at how you build the MDX queries.

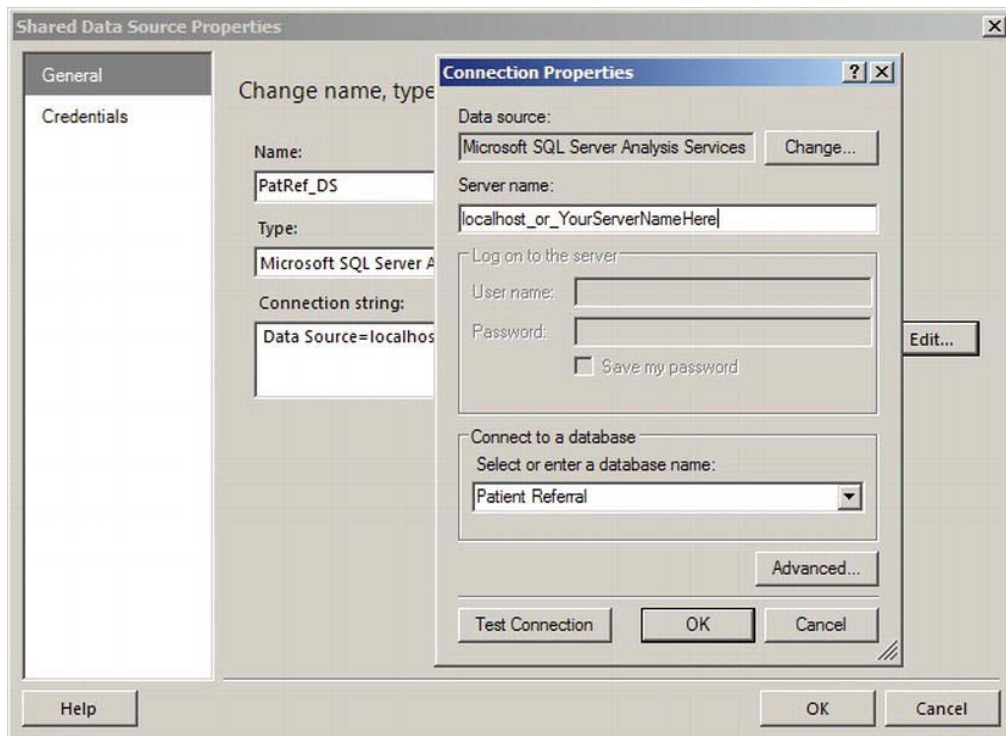


Figure 12-6. Connection properties of the PatRef_DS data source

Working with the Graphical MDX Query Builder

One of the great new features introduced in SSRS for SQL Server 2005, and enhanced further in 2008, is the graphical MDX query builder. MDX is a fairly large, complex language that is prone to syntactical errors. You must innately understand it to truly deliver precision data from SSAS cubes. Having and using a graphical query tool to form the base MDX query, in much the same way that developers use the graphical query designer for T-SQL, reduces common syntax errors and speeds development of the query.

With the Average Referral to Admission report open to the Report Data Query window, notice that you have several dimensional and measure elements listed: BranchName, Year, MonthName, Time to Admission, and Fact Referral Count. By default the report will open in the graphical design mode, as shown in Figure 12-7, and not the generic query designer. The design modes are toggled with the Design Mode button on the right of the toolbar. In the graphical design mode, the dimensional elements are dragged and dropped directly in the window from the Metadata pane where the elements are listed. If the Auto Execution button is selected, which it is by default, whenever elements are dragged and dropped, the query executes, and the data results are displayed.

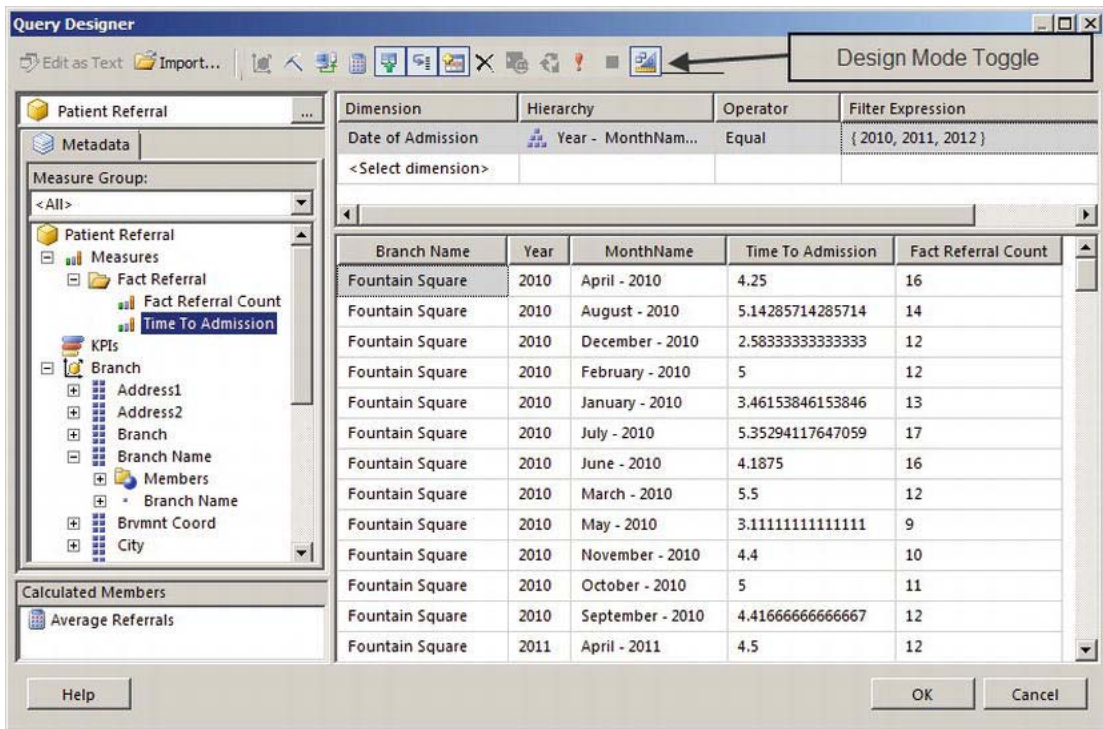


Figure 12-7. Output from graphical query builder

Like the multidimensional view of the data, as you saw when you queried the same cube in the Patient Referral cube browser, SSRS sees the data two-dimensionally, with rows and columns. An SSRS report can access data from a cube; however, it is necessary to arrange the data two-dimensionally because this is how the report will aggregate the data when it sums and totals the measured values, in this case Time to Admission and Fact Referral Count. We have also applied a filter to the query so that only the years from 2010 through 2012 will be selected.

As you may have noticed in the bottom left corner of Figure 12-7, you can use several buttons on the toolbar to modify the properties of the query, such as showing empty cells or adding calculated members. A calculated member is part of the MDX query that is created by combining one or more elements into a value that can be used independently as a new element. You could, for example, create calculated members to not only show the average time to admission for each dimensional element such as BranchName and Year, but also show the minimum time to admission, using the MIN function, or the maximum time to admission, using the MAX function. The calculated members would become part of the overall query that could be used as new measures with data values returned at each dimensional level. In other words, if you had to create a calculation at the report layer that wasn't created as a calculation in the cube itself, you could create a Calculated Member at the dataset layer of the report. For those not skilled in writing MDX calculations, this is an easy way to create report level calculations.

With the graphical query built and working, let's now look at the MDX that was created behind the scenes. You can do this by clicking the Design Mode toolbar button. Figure 12-8 shows the MDX query that returns the selected dimensional elements and measures for the report.

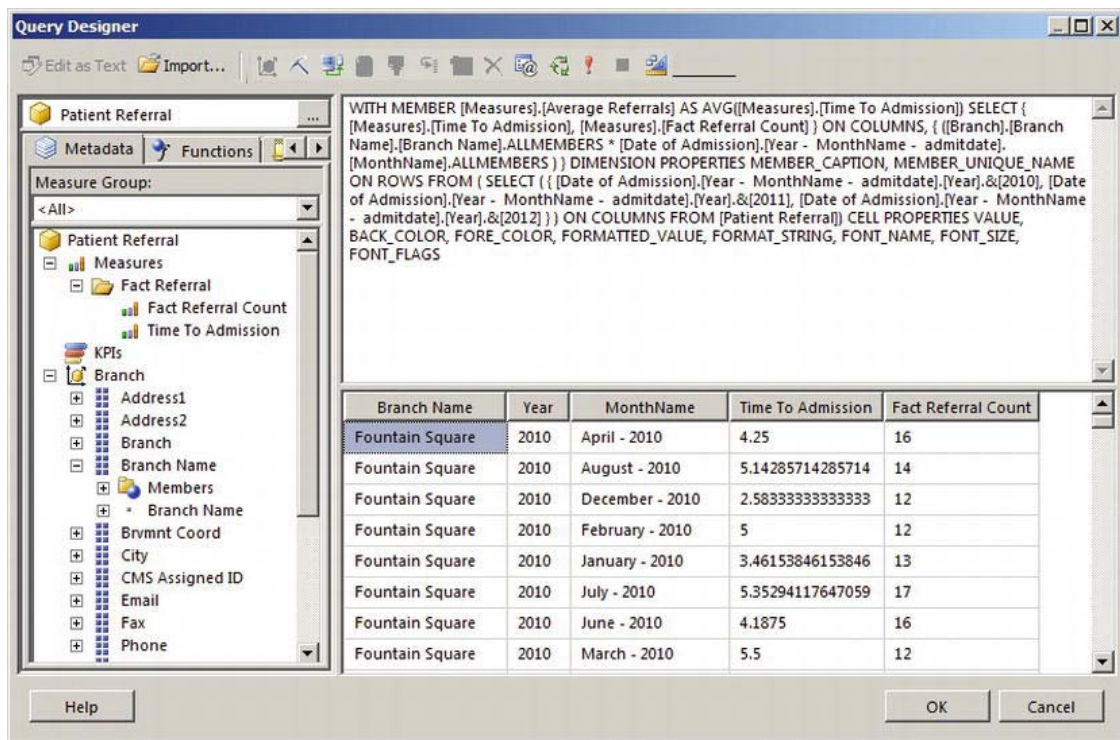


Figure 12-8. MDX query generated graphically

■ **Caution** If you modify the MDX query while in text mode and try to return to design mode, you could potentially lose any changes you made to the query manually, as shown in Figure 12-9.

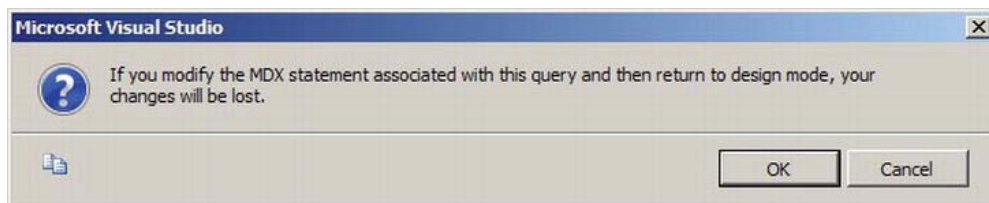


Figure 12-9. Warning message displayed when altering MDX query

Both reports, Average Time to Admission and Referral to Admission Chart, will be displayed in a compact Web Part in SharePoint, which we will show how to do in the next section. Because of this, we

intentionally made the size of the reports small, with 8-point Arial font and narrow cells. Figure 12-10 shows the Design tab for the Average Time to Admission report.

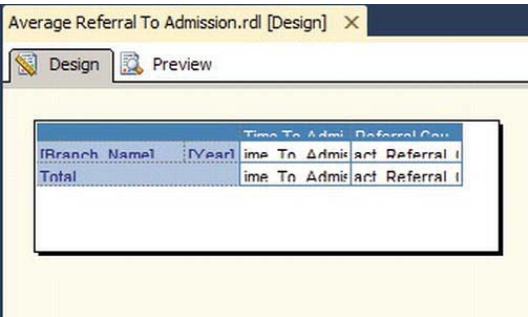


Figure 12-10. Layout tab for Average Referral to Admission report

At this point, we could publish both reports to the reporting services instance and deliver the reports to users. These two reports combine a level of BI that is perfect for a view within a dashboard or business portal. To that end, we will take a final view of the second chart-based report in the design environment and then move on to a fully integrated Microsoft SharePoint deployment, where the reports can be viewed and managed within SharePoint itself. Figure 12-11 shows the Referral to Admission Chart report previewed in BIDS.

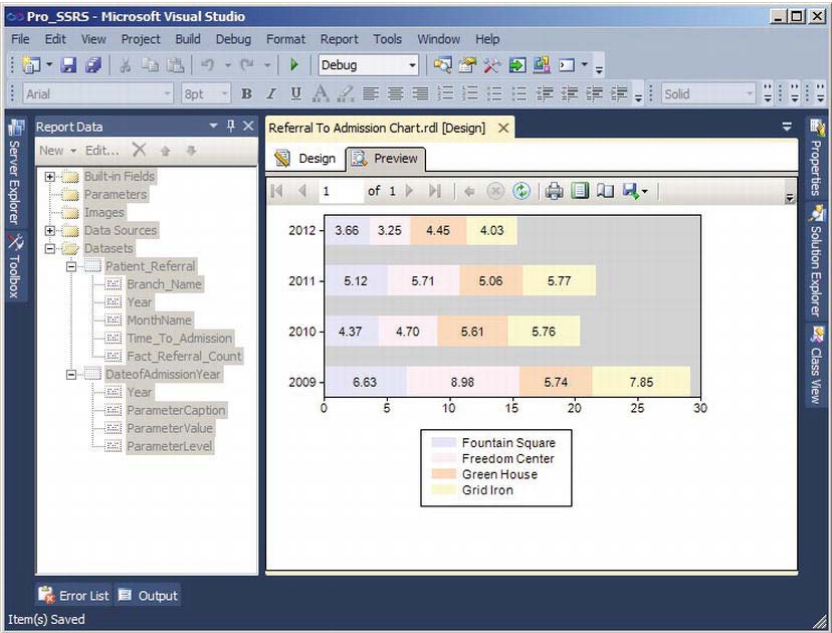


Figure 12-11. Previewing Referral to Admission Chart in the design environment

Incorporating SSRS with Microsoft SharePoint 2010

SSRS and SharePoint technologies have progressed and matured similarly. Both have enjoyed growing acceptance and support from businesses around the world. It was a natural progression, therefore, for the two technologies to one day come together to form components of the overall Business Intelligence platform that Microsoft has built over the past several years. Previous support for SSRS in SharePoint included SSRS Web Parts, which could be installed for SharePoint installations. These Web Parts provided a means to view published SSRS reports within SharePoint. However, this union was not as adhesive as many administrators and dashboard designers needed it to be. Management of reports still fell to the likes of DBAs or report content managers. SQL Server 2005 Service Pack 2 included support for a fully integrated SSRS and SharePoint infrastructure, whereby reports could be published, managed, and viewed all within SharePoint. SSRS 2008 and 2008 R2 continued this merger, and many organizations have adopted this methodology over a native-only SSRS that does not integrate directly with SharePoint.

This brings us to SharePoint 2010 and SQL Server 2012 Reporting Services. Although similar to previous versions in integrating Reporting Services with SharePoint, the installation and configuration process is very much different. In the latest release of Reporting Services, if you wish to run in SharePoint integrated mode, you will need to install SharePoint and then install Reporting Services in SharePoint integrated mode. This process essentially installs Reporting Services on top of SharePoint 2010. This process differs from prior editions, in that SSRS and SharePoint were separate entities and could be integrated using a configuration tool.

In this section, you will walk through a SharePoint and SSRS stand-alone installation and deploy to and view on the SharePoint site the two reports from the previous section, Average Time to Admission and Referral to Admission Chart:

1. First, you will install Microsoft SharePoint 2010 with Service Pack 1 and SSRS 2012 on the same server.
2. Next, you will deploy reports to the SSRS-integrated server with SharePoint 2010.
3. Finally, you will create a simple dashboard using SSRS Web Parts.

Installing SharePoint 2010 and SQL Server 2012 on a Stand-Alone Server

The installation routines for SQL Server 2012 have been greatly enhanced over the years. Specifically, with SSRS, you maintain the ability to install the core assemblies, but configure the SSRS instance later. However, in SSRS 2012, you can install SSRS in SharePoint integrated mode directly if you have planned to roll out SSRS and SharePoint together. For our installation, we chose to install both SSRS 2012 and SharePoint 2010 on the same server, known as a stand-alone installation. This is the recommended setup for anyone who wants to familiarize themselves with how the two technologies work together. Other deployment scenarios, such as multiple-server installations, are also possible and expected in larger environments, but we chose the stand-alone deployment initially solely for the sake of testing and training. We would recommend this path for anyone not already familiar with an integrated SSRS and SharePoint installation.

■ **Note** As with the majority of server-based software, SharePoint 2010 and SQL Server 2012 have prerequisites that must be met before a successful installation occurs. These requirements can change over time, so it is always best to verify the prerequisites before setting out to install the software. For example, SharePoint 2010 needs to be installed on a server-based operating system like Windows Server 2008. It also requires Service Pack 1 to be installed and a minimum of 4GB of ram and 80 GB of disk space. The server must be configured as a Domain Controller. Those are just a few of the requirements, but you can see more SharePoint requirements at <http://technet.microsoft.com/en-us/library/cc262485.aspx> and SQL Server Reporting Services requirements at [http://msdn.microsoft.com/en-us/library/gg492276\(v=sql.110\).aspx](http://msdn.microsoft.com/en-us/library/gg492276(v=sql.110).aspx).

Assuming that all of the prerequisites, software and hardware requirements have been met then there are six major steps to complete a stand-alone installation of SSRS and SharePoint:

1. Install SharePoint 2010 with Service Pack 1.
2. Install SQL Server 2012 Database Engine and Reporting Services in SharePoint mode.
3. Configure SharePoint 2010.
4. Install and start the Reporting Services SharePoint Service.
5. Create a Reporting Services Service Application in SharePoint Central Administration.
6. Configure Reporting Services Integration with SharePoint.

Installing SharePoint 2010

We will start by walking you through the installation of SharePoint 2010 with Service Pack 1. Installing SharePoint 2010 is a straightforward process, with the exception of the final step, where the installation wizard asks you if you want to Run Configuration Wizard (Figure 12-12). This option button is selected by default, but in the case of Reporting Services in SharePoint integrated mode, you will configure it later. You'll configure SharePoint after you have installed an instance of the SQL Server 2012 database engine. Remove the check from the checkbox and click Close to complete the SharePoint 2010 installation.

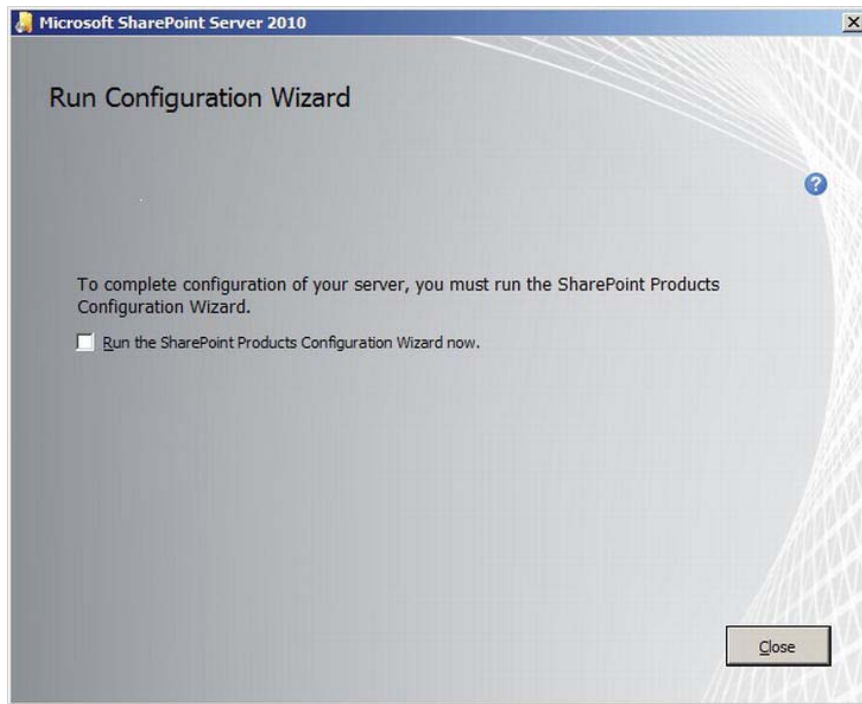


Figure 12-12. Installing SharePoint 2010

Installing SQL Server 2012 Reporting Services in SharePoint Mode

Now that SharePoint 2010 is installed, it is time to install the Database Engine and Reporting Services for SharePoint. The SQL Server 2012 installation with SSRS is a standard procedure that you may already be familiar with. Going through each step of the installation is outside of the scope of this book; however, the only difference between a typical install and one that includes support for integrated SharePoint can be seen on the Feature Selection and the Reporting Services Configuration screens. Go ahead and choose the options shown in Figure 12-13.

Since this is being done on a local VM, not on a traditional corporate network, I chose to install Analysis Services on this VM as well. That way, I can copy the Pro_SSRS project and deploy the Patient Referral Analysis Services database (the cube) to the same VM for the remaining examples in this chapter.

■ **Note** If you are also performing the SharePoint 2010 installation on a dedicated VM, you also will need to copy your Pro_SSRS project to the VM and deploy the cube as discussed previously in this chapter.

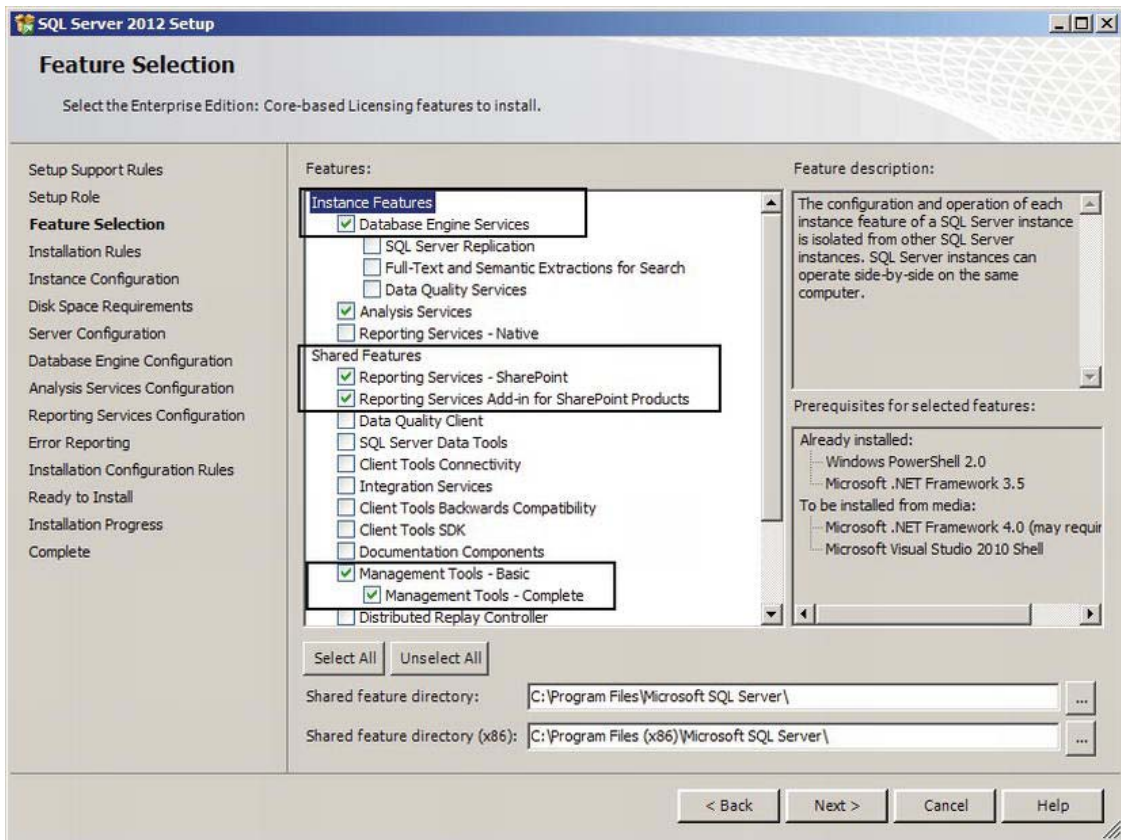


Figure 12-13. Installing SSRS for SharePoint

As shown in Figure 12-14, the only option that you have is Reporting Services SharePoint Integrated Mode – Install Only. In the note under the Install Only radio button, you can see a message stating that you will need to configure SharePoint using Central Administration to complete the installation. It also states that you'll need to get the Reporting Services service started and create a Reporting Services service application (installation steps 4 and 5 in the previous list). Now, you will install the Reporting Services SharePoint Service.

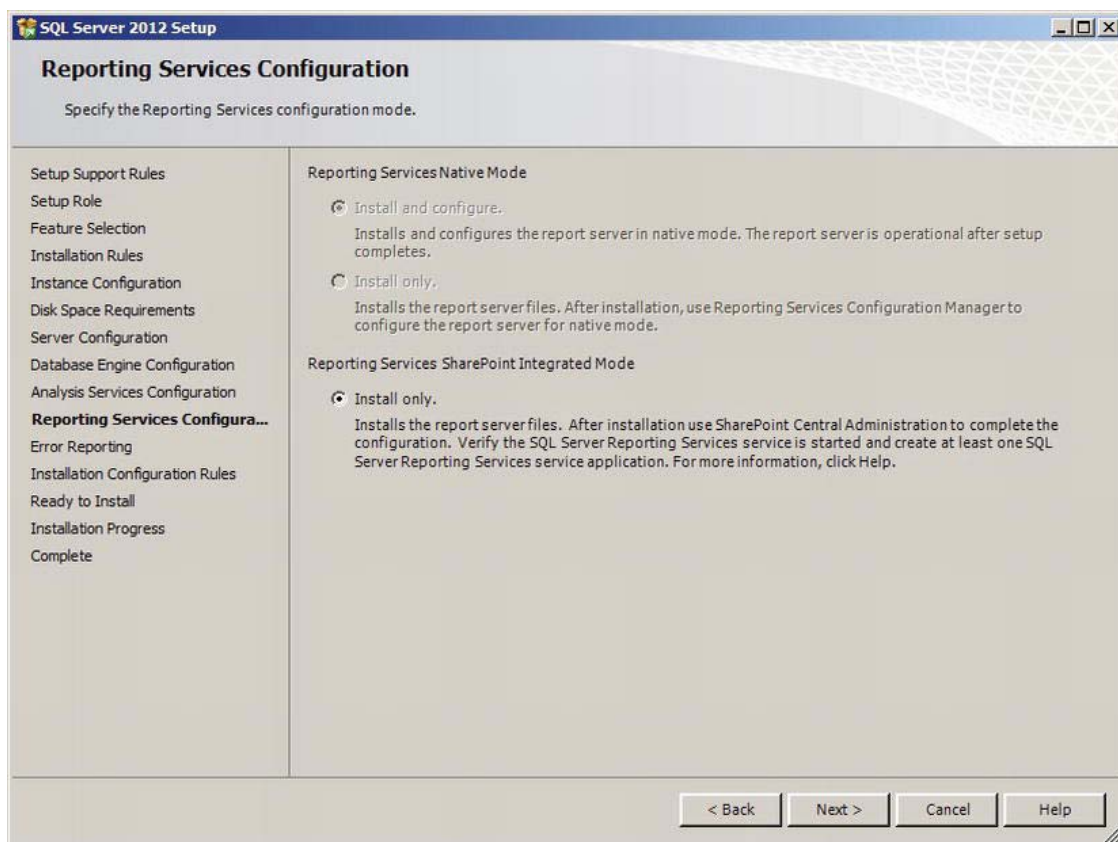


Figure 12-14. Installing SSRS in SharePoint Integrated Mode

Configuring SharePoint 2010

Once the SSRS installation is complete, you need to perform the third major step in the installation process. Now let's configure SharePoint using the SharePoint 2010 Products Configuration Wizard under the Microsoft SharePoint 2010 Products folder of the All Programs menu. Follow the below steps to configure your new server farm.

1. Choose the option to Create a new server farm and click Next to proceed.
2. Specify the Configuration Database Settings for the SharePoint site as shown in Figure 12-15. In this case, my server name is VMWINSVR2008SP2 and I specified the user credentials (SQLBIGEEK\bmcDonald) to use when connecting to the configuration database. Normally, you would specify a system account with a password that rarely changes. Click Next to proceed.
3. Specify a Passphrase to use for the Farm Security Settings. This is used when you need to perform operations like joining servers to the farm. The minimum

requirement for this is like that of a secure Active Directory network. The password must have the following characteristics: a minimum length of eight characters, special characters, and a mixture of numerals and capital and lower-case letters. Click Next to proceed.

SharePoint Products Configuration Wizard

Specify Configuration Database Settings

All servers in a server farm must share a configuration database. Type the database server and database name. If the database does not exist, it will be created. To reuse an existing database, the database must be empty. For additional information regarding database server security configuration and network access please see [help](#).

Database server:


Database name:

Specify Database Access Account

Select an existing Windows account that this machine will always use to connect to the configuration database. If your configuration database is hosted on another server, you must specify a domain account. Type the username in the form DOMAIN\User_Name and password for the account.

Username:

Password:



< Back Next > Cancel

Figure 12-15. Specify configuration database settings for SharePoint

4. Next, you'll need to configure the SharePoint Central Administration Web Application. I like to change the default port number to something that is not commonly used, but a little easier to remember than the typical default. Feel free to keep the default port, but I chose to use port 1212, as shown in Figure 12-16. Click Next to proceed.
5. After completing the wizard, you will be presented with a summary. Click Next to begin the configuration process. After the configuration wizard is done working its magic, you will be prompted with a dialog box stating that the process has been completed. Upon clicking Finish, you are presented with the SharePoint Central Administration. If for some reason yours does not open up, you can find it by going to the Start Menu, All Programs, and then Microsoft SharePoint 2010 Products. Alternatively, you could fire up Internet Explorer

and navigate to `http://ServerName:PortNumber/`. In my scenario, running on server name `VMWINSVR2008SP2` on port 2012, my URL would be `http://VMWINSVR2008SP2:2012/`.

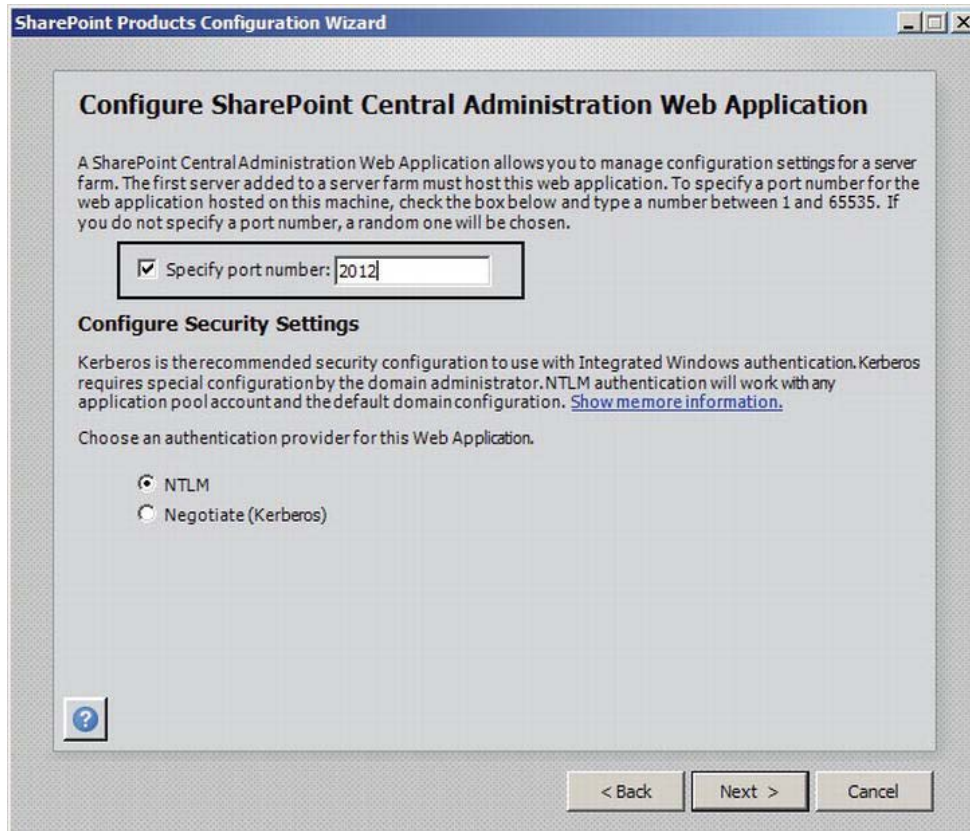


Figure 12-16. Specify SharePoint Central Administration port and security

Whew! You're done with the configuration wizard. Now comes the fun part. I bet you didn't think that you would be learning about PowerShell, did you? Well, you won't be learning too much, so don't be concerned. Now, you'll perform the fourth major step in getting Reporting Services 2012 and SharePoint 2010 to work together.

Installing and Starting the Reporting Services SharePoint Service

In this step, you need to connect the installed SQL Server Reporting Services components to your new SharePoint farm. Perform the following steps to install and start the Reporting Services SharePoint Service.

Click on the Start button, All Programs, and then right-click and select Run as administrator on the SharePoint 2010 Management Shell under the Microsoft SharePoint 2010 Products folder. You have to

run this as an administrator to perform the following steps. If not, you will receive the error displayed in Figure 12-17.

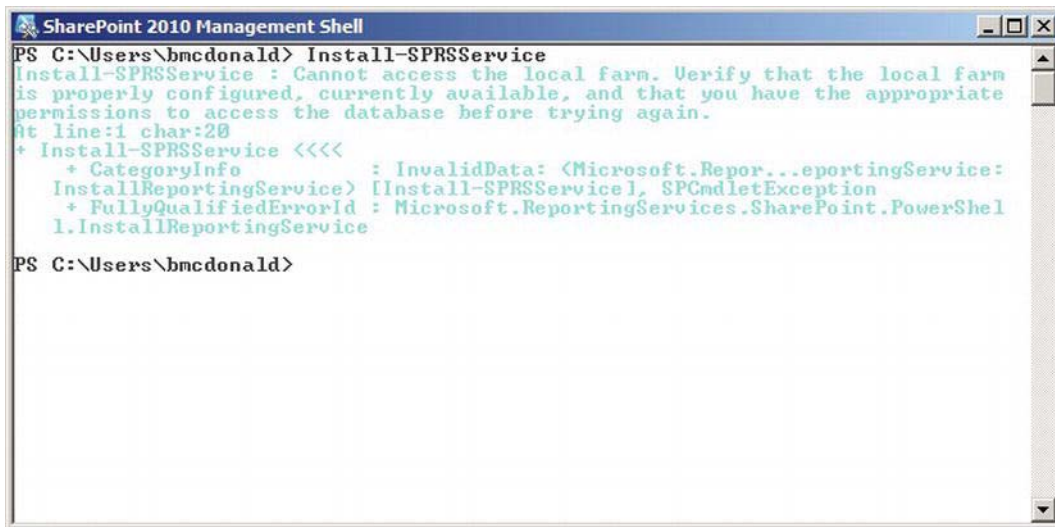


Figure 12-17. Error if SharePoint management shell not in administrator mode

Type in the following PowerShell command to install the SharePoint Reporting Service. Hit the Enter key after typing in the command to execute it.

Install-SPRSService

Type in the following PowerShell command to install the PowerShell service proxy. Hit the Enter key to execute the command.

Install-SPRSServiceProxy

Type in the following PowerShell command to start the service. After typing the command on one line, hit the Enter key to execute the command.

Get-spserviceinstance -all | where {\$_.TypeName -like "SQL Server Reporting"} | Start-SPServiceInstance*

Figure 12-18 shows the result after completing the steps installing and starting the Reporting Services SharePoint Service. For the fifth step, you need to create a new Reporting Services Service Application.

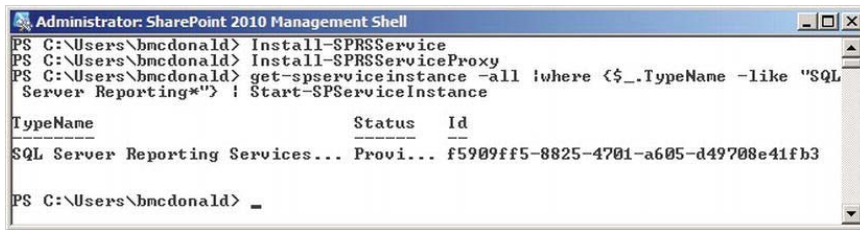


Figure 12-18. SharePoint management shell and scripts in administrator mode

Creating a New Reporting Services Service Application

You are now on the cusp of having all of the setup steps required to install Reporting Services 2012 in SharePoint 2010 mode. This step will be performed in SharePoint 2010 Central Administration. Perform the below steps to create a new SQL Server Reporting Services Service Application. I know, it's a mouthful, but we'll get through this together.

You need to open up SharePoint 2010 Central Administration, which can be found under the Start menu, All Programs, and then Microsoft SharePoint 2010 Products.

From the Central Administration home screen, click on the Manage Service Applications link under the Application Management task group as pointed out in Figure 12-19.

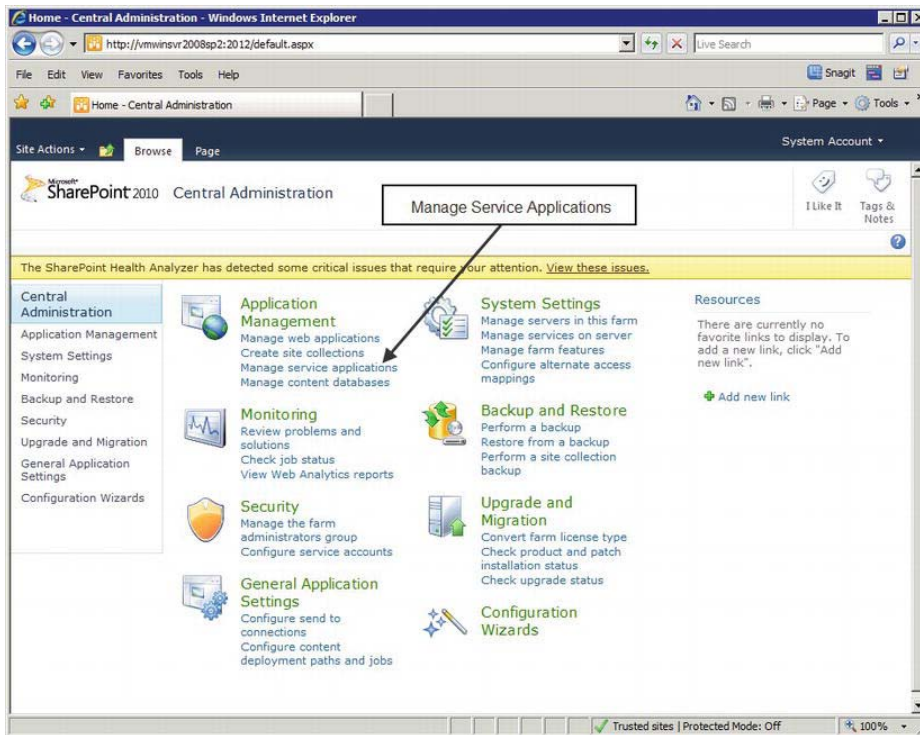


Figure 12-19. SharePoint 2010 Central Administration manage service applications

Next, click the New button in the SharePoint menu and select SQL Server Reporting Services Service Application as shown in Figure 12-20.

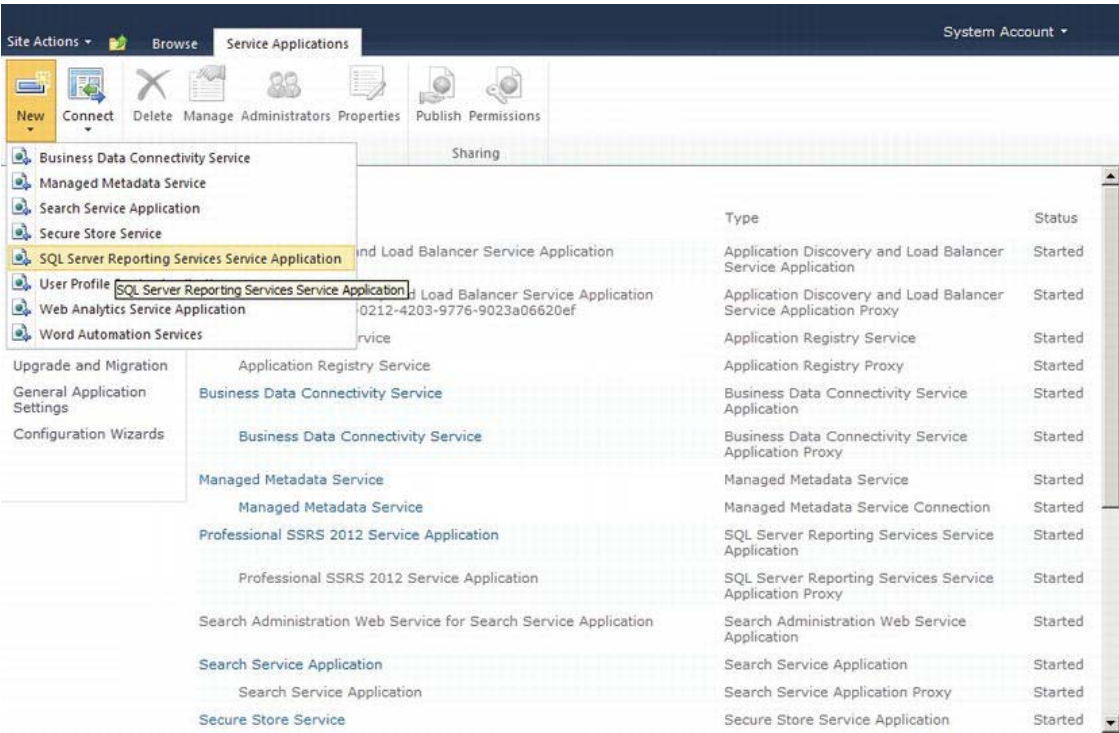


Figure 12-20. Creating a new SSRS Service Application

Enter a name and Application Pool to be used for this Service Application. From the manageability perspective, it is a good idea to give them both the same name. As can be seen in Figure 12-21, mine is named Professional SSRS 2012 Service Application for both. Depending on your security needs, change appropriately. I used the default security settings for the SharePoint managed account. Scroll down and set the database server, name, and authentication details appropriately. Last but not least, choose the Web Application Association. In this case, there is only one web application to associate with. Click OK to save your settings. After a few minutes, you should see a screen stating that your new service application has completed successfully.

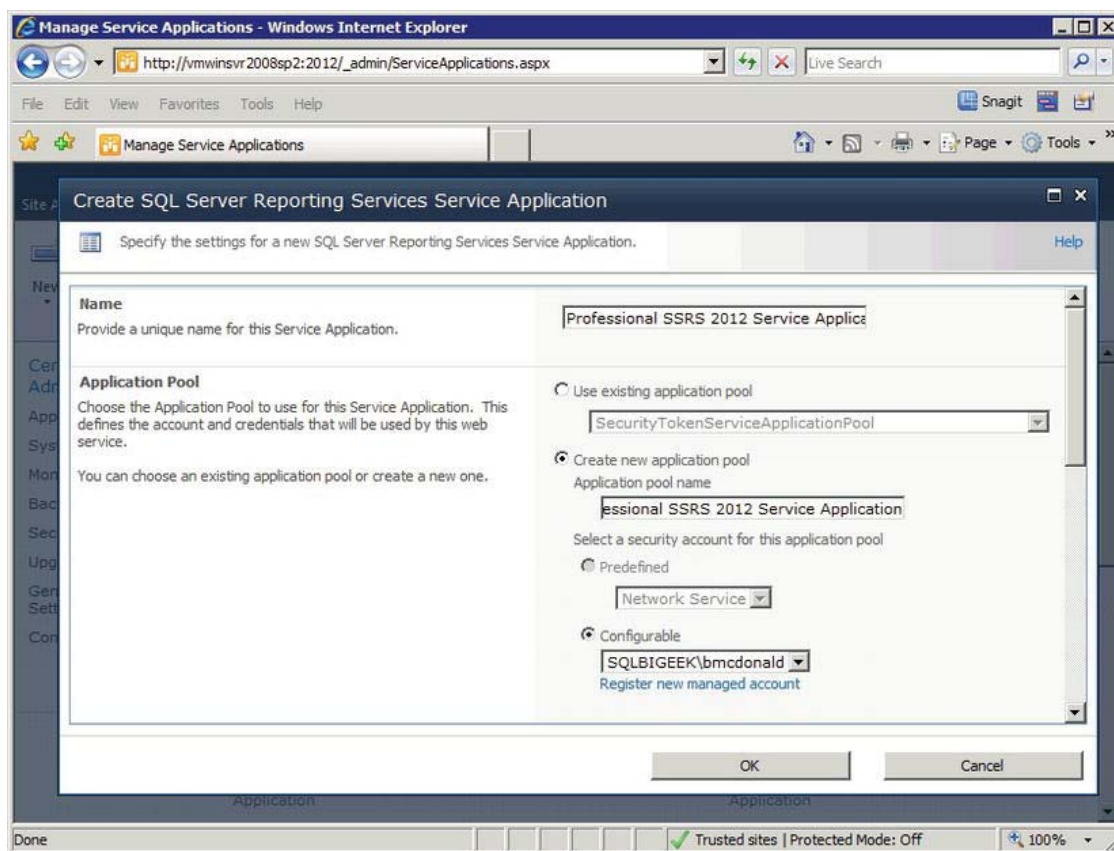


Figure 12-21. Creating a new SSRS Service Application

After the new SQL Server Reporting Services Service Application has been created, you will see it in your list of Service Applications as shown in Figure 12-22. I'm sure that you will be happy to hear that you only have one more step left in your installation and configuration of SQL Server 2012 Reporting Services in SharePoint 2012 mode.

In prior versions of Reporting Services, integration with SharePoint was managed using the Reporting Services Configuration Manager. As I mentioned earlier, in SQL Server 2012, if you choose to run in SharePoint integrated mode, Reporting Services is installed and configured completely within SharePoint 2010. So next, I'll walk you through the steps to get the Reporting Services integrated into SharePoint.

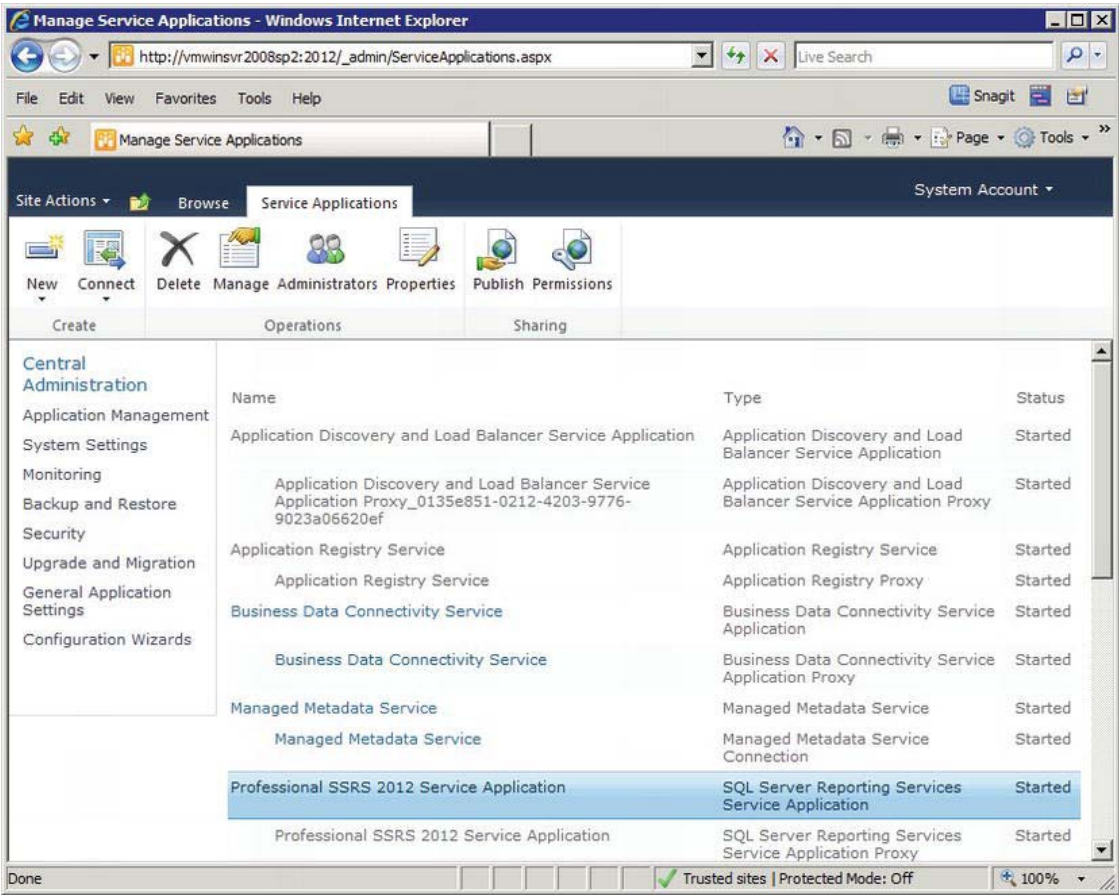


Figure 12-22. List of service applications

Configuring Reporting Services Integration with SharePoint

Now that you have completed all of the required installation steps, it is time to begin the final configuration process. You know that SSRS is already configured for SharePoint integration. All that is required now is to configure SharePoint to activate the Reporting Services Integration and Administration Features.

Now that we are in SharePoint integrated mode using SharePoint 2010 and SQL Server 2012, the ReportServer database is managed completely by SharePoint. SharePoint will supply the storage for all of the security, reports, and data sources. All of the benefits of SharePoint, such as document publishing and delivery, will be maintained by SharePoint.

SharePoint's main administration Web site, called Central Administration, is where you will perform the remaining SSRS configuration settings. Perform the following configuration steps:

1. Open the Central Administration Web site by clicking Start, All Programs, Microsoft SharePoint 2010 Products, and then SharePoint 2010 Central Administration.
2. Click on Site Actions and then Site Settings. When the Site Settings screen appears, click on the Site Collection Features link under Site Collection Administration. From there we need to ensure that the Reporting Services features are activated.
3. Click Activate next to the Report Server Central Administration and the Report Server Integration features as shown in Figure 12-23.

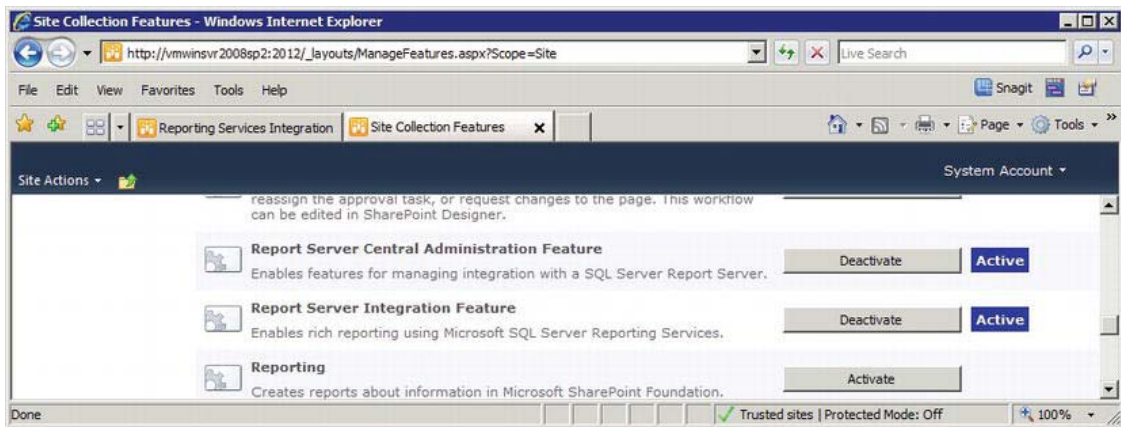


Figure 12-23. *Activating site collection features*

That's all there is to it! You're done with the installation and configuration of SharePoint 2010 and SQL Server 2012 Reporting Services in SharePoint mode. Before we move on, though, I want to show you how to get to the screen for administering your Reporting Services configuration settings. If you still have Central Administration open, click on Application Management. Next, click on Manage Service Applications under the Service Applications group. You should recognize the screen that opens, as it is the same one displayed in Figure 12-22 above. Drill into your Reporting Services Service Application, as I have done in Figure 12-24. Remember that mine is Professional SSRS 2012 Service Application.

■ **Note** Different environments may call for different deployment circumstances and methodologies. Should you need further assistance with getting SSRS 2012 and SharePoint fully integrated, there are some great resources available at Microsoft.com. One such resource for configuring SSRS 2012 and SharePoint 2010 can be found at [http://msdn.microsoft.com/en-us/library/gg492276\(v=sql.110\).aspx](http://msdn.microsoft.com/en-us/library/gg492276(v=sql.110).aspx)

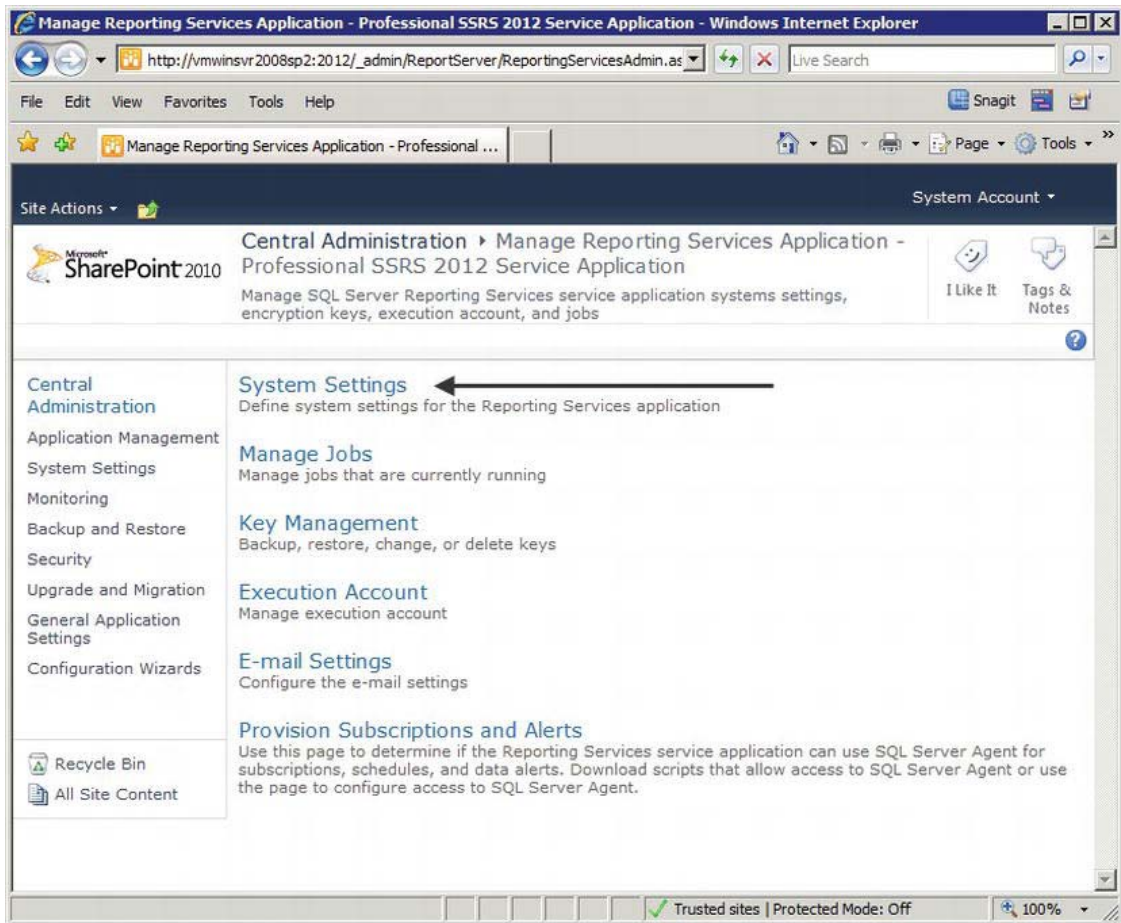


Figure 12-24. Managing Reporting Services service system settings

Next, click on the System Settings link to view the Reporting Services Service settings. As you can see in Figure 12-25, this is where you can change various options like report history, timeouts, and even how long to save the Execution Log as we did in Chapter 10.

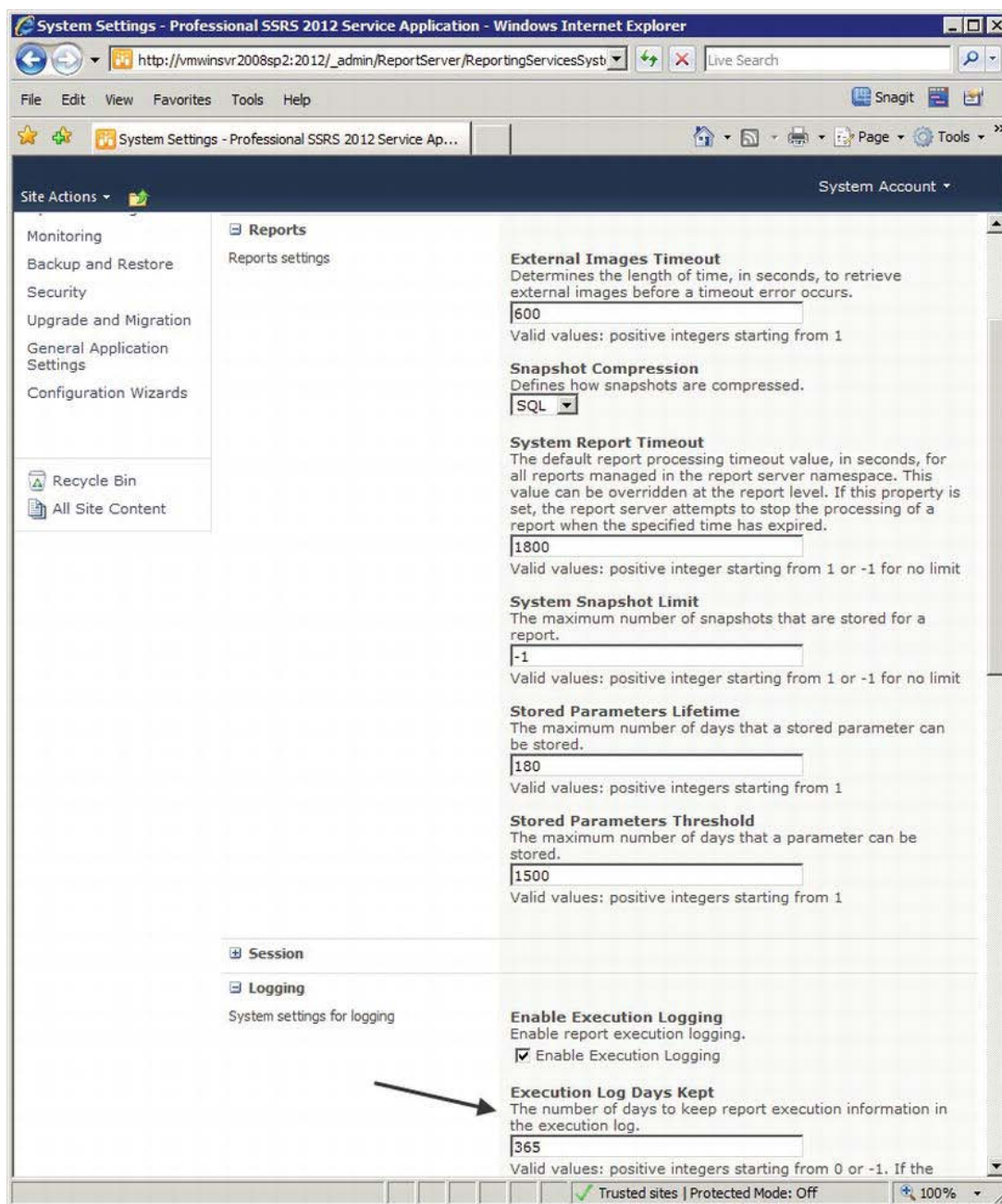


Figure 12-25. Setting Reporting Services Service Application server defaults

Now that you are done with all of your setup and configuration, it is time to move forward and show you how to deploy reports to a SharePoint site.

Deploying Reports in a SharePoint-Integrated SSRS Installation

Once the three management tasks have been completed, it is time to begin working within the SharePoint site itself, where you will work with deployed reports and dashboards to create a portal page for your users. In our organization's case, we wanted a simple Admissions dashboard page where company executives and decision makers could review important BI reports concisely. SharePoint provides many templates and samples that you can use to aid you in designing your dashboard.

Let's take a look at a default SharePoint page. You can see in Figure 12-26 that a base SharePoint site collection installation includes a Document Center. To centralize our deployment location, we created a Reports Library. We also created another site on our SharePoint system using a built-in SharePoint template called Business Intelligence Center, which will allow us to use web parts to create a sample dashboard.



Figure 12-26. SharePoint Document Center main page

Figure 12-27 displays the Business Intelligence Center for SharePoint 2010. The name Business Intelligence Center should not lead you to believe that this is where all of the SSRS reports will be

deployed. SSRS reports can be deployed to any valid SharePoint path, as you will see shortly. The Business Intelligence Center is a location where you can create a dashboard or Microsoft Excel workbook as well as an area where reports can be deployed. It is in the Reports Library, which resides within the Document Center, where SSRS RDL files, data sources, and many other types of reports can be stored.

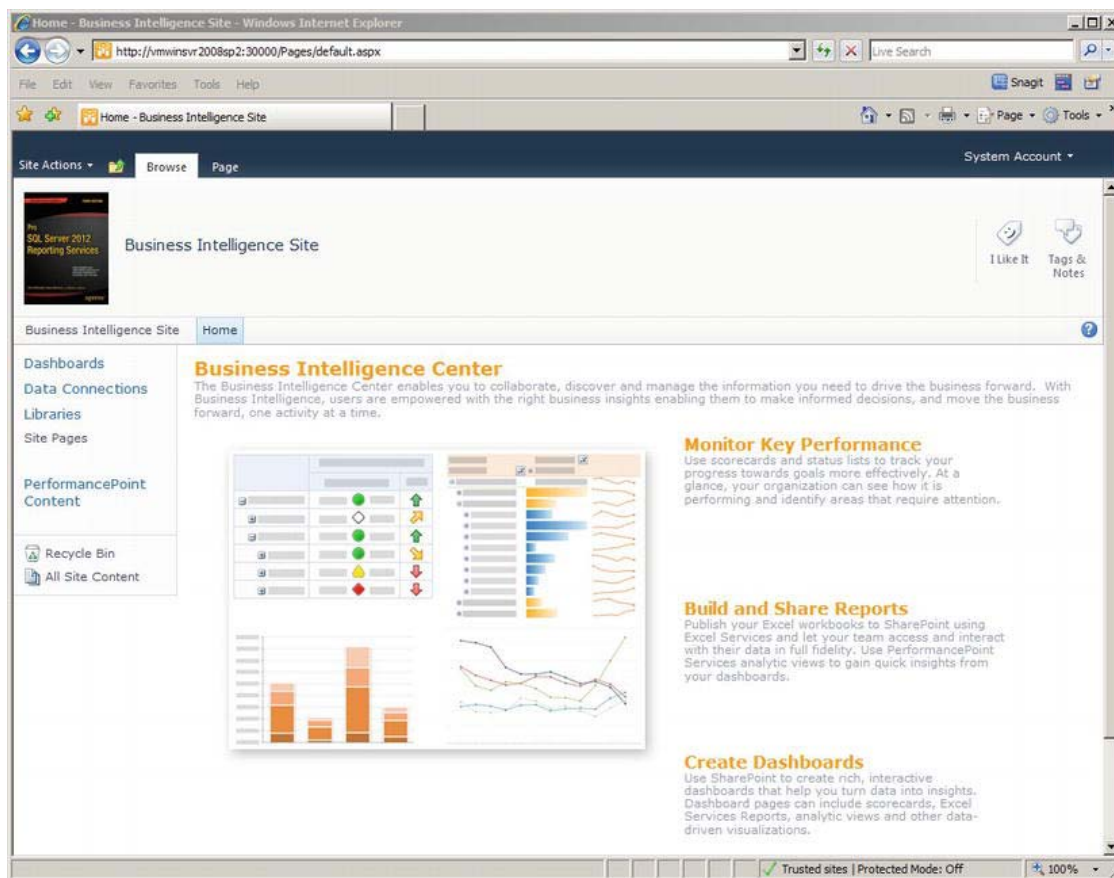


Figure 12-27. *SharePoint Business Intelligence Center*

As I mentioned, RDL reports can be deployed to any valid document library in SharePoint. In Figure 12-28, you can see project properties for your Pro_SSRS report solution. Since you will be deploying these reports to the SharePoint server, the target URLs for the server and report folder will be different from those for a native SSRS deployment. You can see that the server will be a URL to the main SharePoint site, in our case <http://localhost/>. The target report folder is a URL that points to a valid SharePoint document library; in this case, http://localhost/sites/Pro_SSRS/Reports, which is pointing to our Reports Library contained in the Document Center.

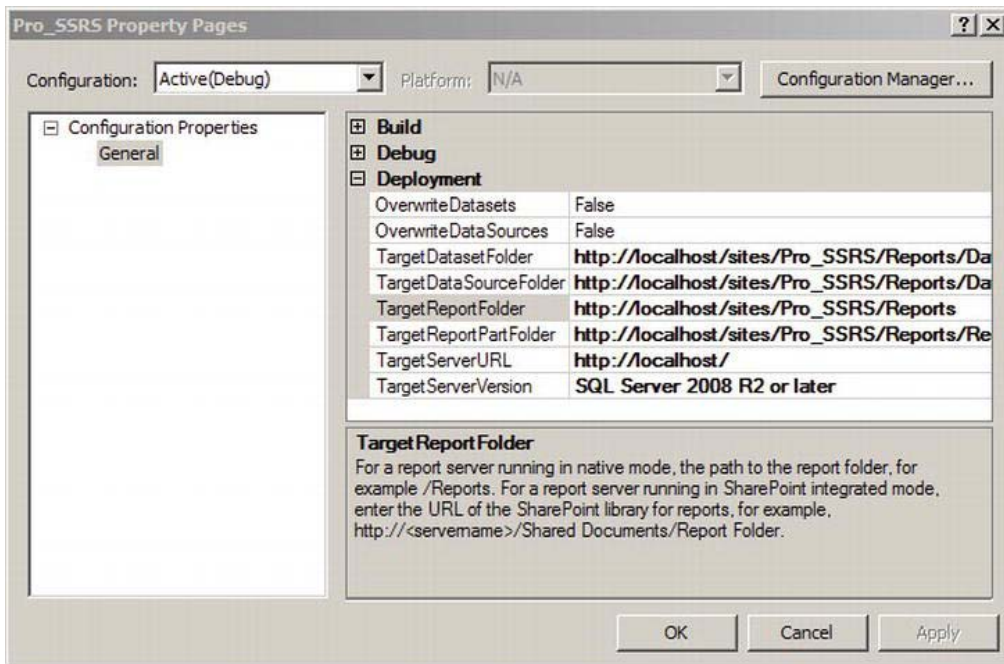


Figure 12-28. Target URL properties for the report project

After you have made these changes to the target URLs for the report project, you can deploy reports directly to SharePoint. Right-click the PatRef_DS Shared Data Source and select Deploy. In order to deploy the report to the SharePoint site, you first must deploy the data source. After the PatRef_DS data source is deployed, right-click the Average Referral to Admission report in the Solution Explorer and select Deploy. Alternatively, you could have selected both the data source and the report by holding the Ctrl key down, then right-clicking and selecting Deploy. This would have deployed the data source first and then the report. Once the report has been successfully deployed, you will be able to see the report within the SharePoint site. The best part about deploying reports to SharePoint, as you can see in Figure 12-29, is that you get to take advantage of all of the SharePoint document features, as RDL files are viewed as documents as well. Features such as workflows, alerts, and document check-in and check-out are inherited.

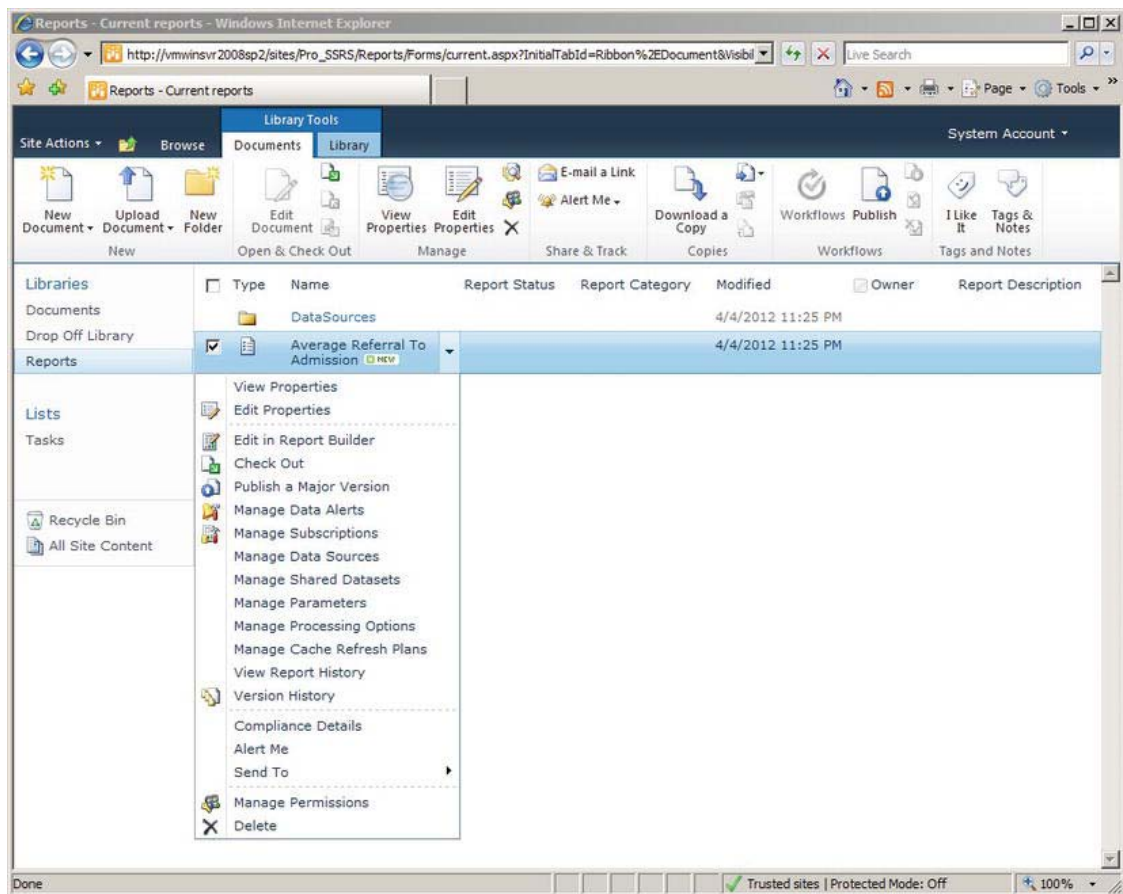


Figure 12-29. Deployed report with SharePoint document features

Before running the report, let's change the PatRef_DS data source to use Stored Credentials, like we did in Chapter 10 when configuring the data source for Subscriptions. We do this because the permissions between SharePoint and the data source may be different than Windows Authentication. If you are having permission-related issues when running a report, this is the first place that I would troubleshoot. Navigate to your deployed data source in SharePoint and alter your data source by clicking on the PatRef_DS link, or clicking the down arrow and selecting Edit Data Source Definition. Select the option to use Stored Credentials under the Credentials section. Enter a valid username and credentials as shown in Figure 12-30. If you are using Windows Credentials, be sure to select the option to Use as Windows Credentials. Click the Test Connection to make sure that you entered valid credentials. If the connection is created successfully, then click OK to save your settings.

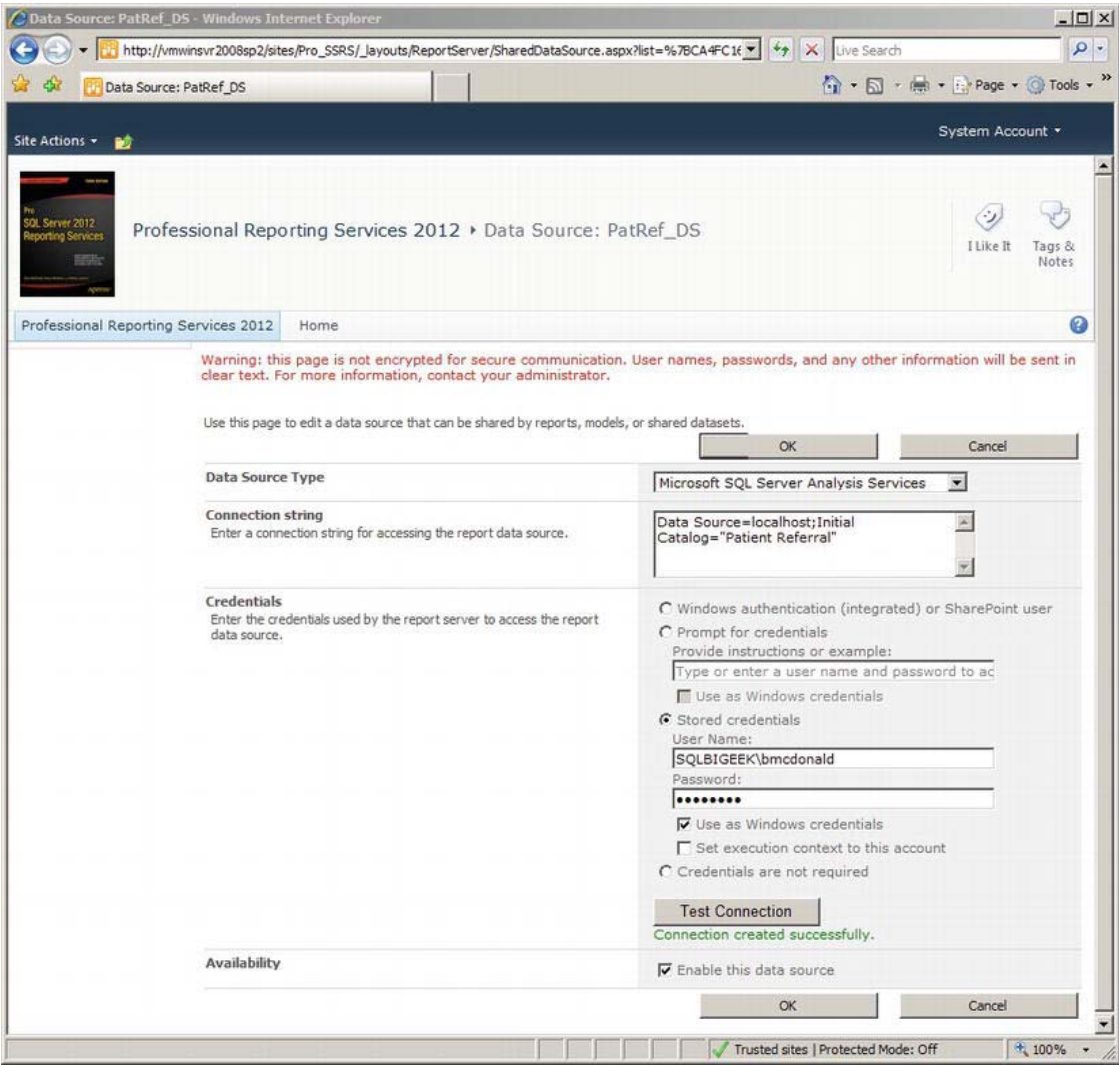


Figure 12-30. Editing data source definition

Next, click the report Average Referral to Admissions. This will open the report in its own page and execute it, much the same way it would be executed via Report Manager. Figure 12-31 shows the report rendered in SharePoint.

		Time To Admission	Referral Count
Fountain Square	2010	4.37	154
	2011	5.12	149
	2012	3.66	52
Freedom Center	2010	4.70	186
	2011	5.71	197
	2012	3.25	52
Green House	2010	5.61	102
	2011	5.06	102
	2012	4.45	66
Grid Iron	2010	5.76	549
	2011	5.77	555
	2012	4.03	200
Unknown	2010		
	2011		
	2012		
Total		5.06	2364

Figure 12-31. Report rendered in SharePoint

You will follow the same deployment steps in BIDS that you did for the Average Referral to Admissions report to deploy the Referral to Admissions Chart report. The processes will be the same, and when complete, you will see the report in the SharePoint web site. Navigate to the report and click it to execute the report. With this report rendered, click on the Actions link button in the top left of the report as shown in Figure 12-32. You may have noticed these actions earlier; but notice the additional actions in the report toolbar, such as Subscribe, Open with Report Builder, and Export. Each of these actions are pretty self explanatory, it is important to realize here that since we are in SharePoint integrated mode, we still have the ability to take advantage of features such as subscriptions and ad-hoc report building. We will cover the Report Builder applications in the next chapter.

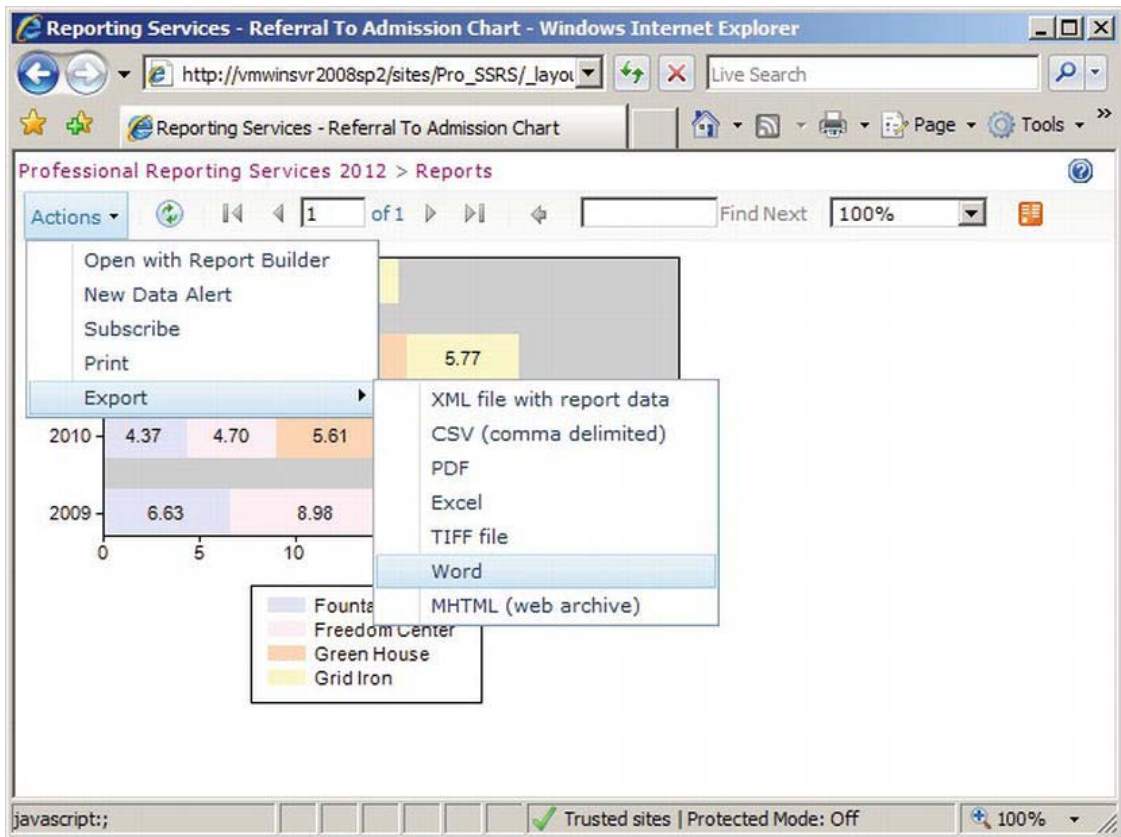


Figure 12-32. Actions available for a report rendered in SharePoint

Creating a Simple Dashboard to Display SSRS Reports

Now that you have successfully deployed and rendered reports in SharePoint and have seen many of the attributes and actions available, I'll show you how to create a dashboard that can render the reports in a concise Web Part. The SQL Server 2012 Reporting Services Add-In that you installed previously included the SQL Server Reporting Services Report Viewer Web Part. Creating a dashboard is as simple as clicking the Dashboards link in the web site and then selecting the New Page option under the Site Actions link button. You will provide a name for the new dashboard link and the initial layout based on a template. For example, I chose to name the dashboard Admissions. Once the dashboard is created, you can navigate to it and edit its design directly in the browser. Click Web Part on the Insert tab while in edit mode and select the SQL Server Reporting Services Report Viewer from the list of available Web Parts, as shown in Figure 12-33. You will find it in the SQL Server Reporting category. After you have it selected, click the Add button.

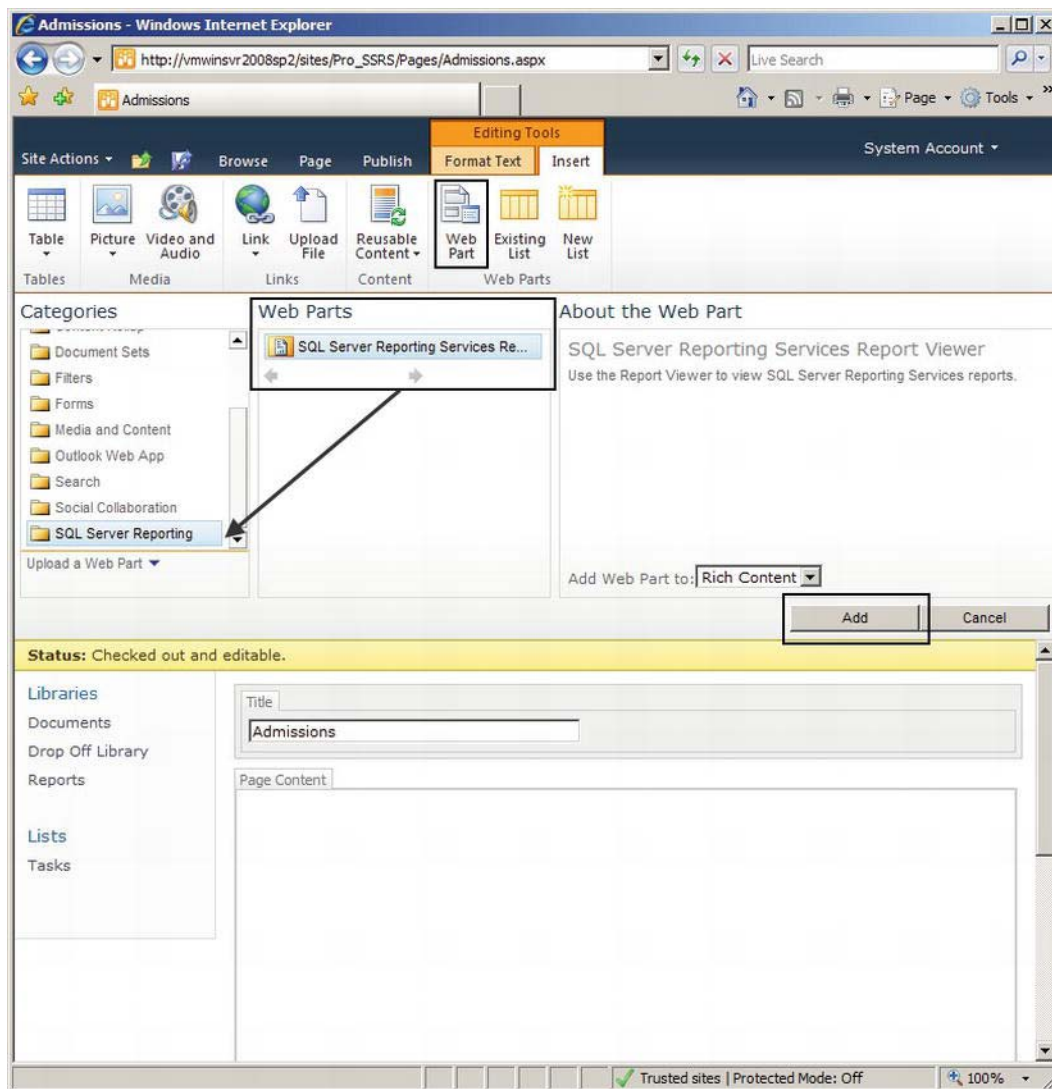


Figure 12-33. Choosing the SQL Server Reporting Services Report Viewer Web Part in SharePoint

Once you add the Web Part to the Admissions dashboard, you will have the ability to edit it to point to a URL where the deployed reports reside. You can click the link to open the tool pane where the report URL will be entered. This links the Web Part directly to the report. You can see the Web Part and the report selection toolbar on the right. In this case, you will select the Referral to Admissions Chart report, which you can see in Figure 12-34. Click Apply to save the Web Part and render the report.

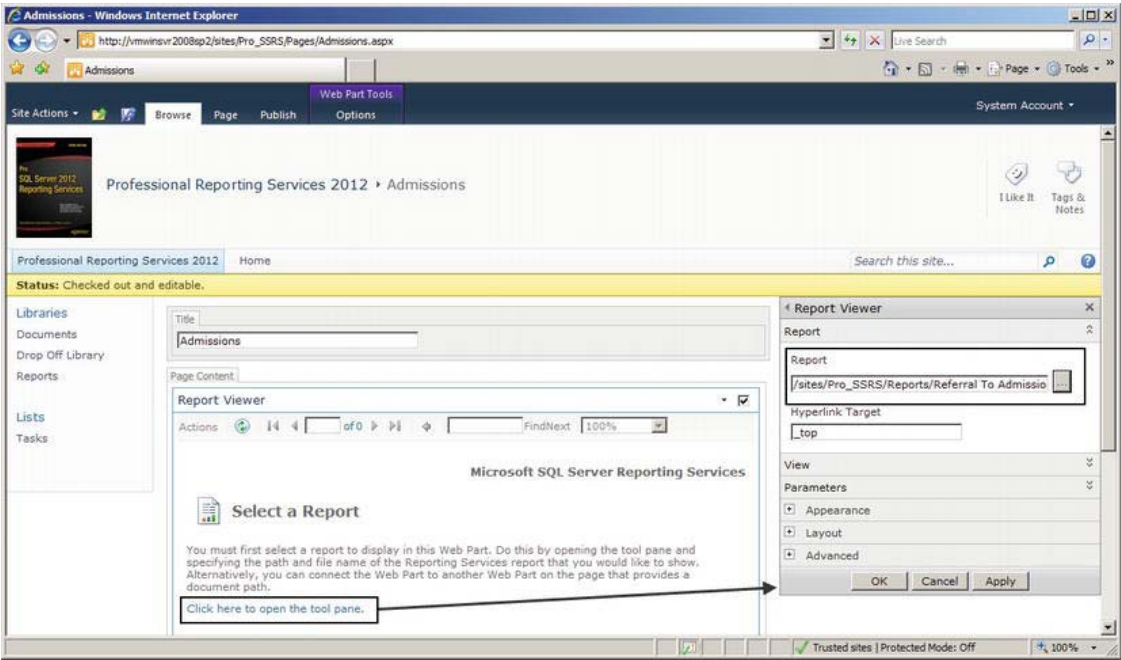


Figure 12-34. Selecting a report URL for the Web Part

Finally, you can view the rendered report in the Web Part that is contained within your dashboard. If you are not happy with the layout—there is too much white space, for example—you can resize the Web Part by navigating back to the Web Part toolbar and entering your desired size settings. Figure 12-35 shows the newly deployed Referral to Admissions Chart report rendered in the Web Part. This basic report can be combined with other SharePoint dashboard Web Parts, like the KPI and Excel Web Access Web Parts, to create an appealing and informative portal for your organization.

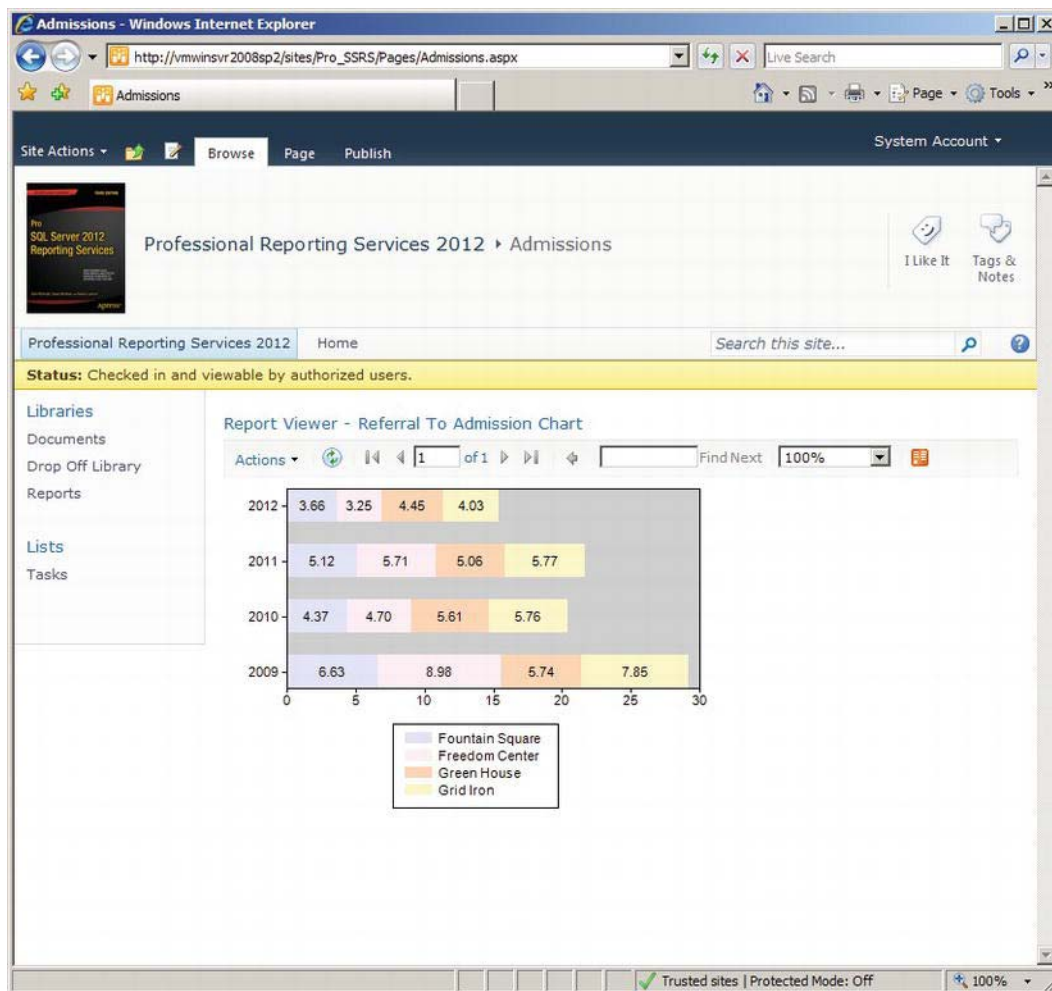


Figure 12-35. Report rendered in Web Part dashboard

Creating Data Alerts

SQL Server Reporting Services 2012 in SharePoint Integrated mode cannot be complete without discussing data alerting. This great new feature gives you the ability to receive alerts when the data contained in a report is changed or when it meets certain pre-defined criteria. The pre-defined criteria are specified when creating a rule within the New Data Alert window. Follow these steps to create a new data alert:

1. If you still have the Referral to Admission Chart report open, click on the Actions button. If not, run the Referral to Admission Chart report as shown in Figure 12-36.

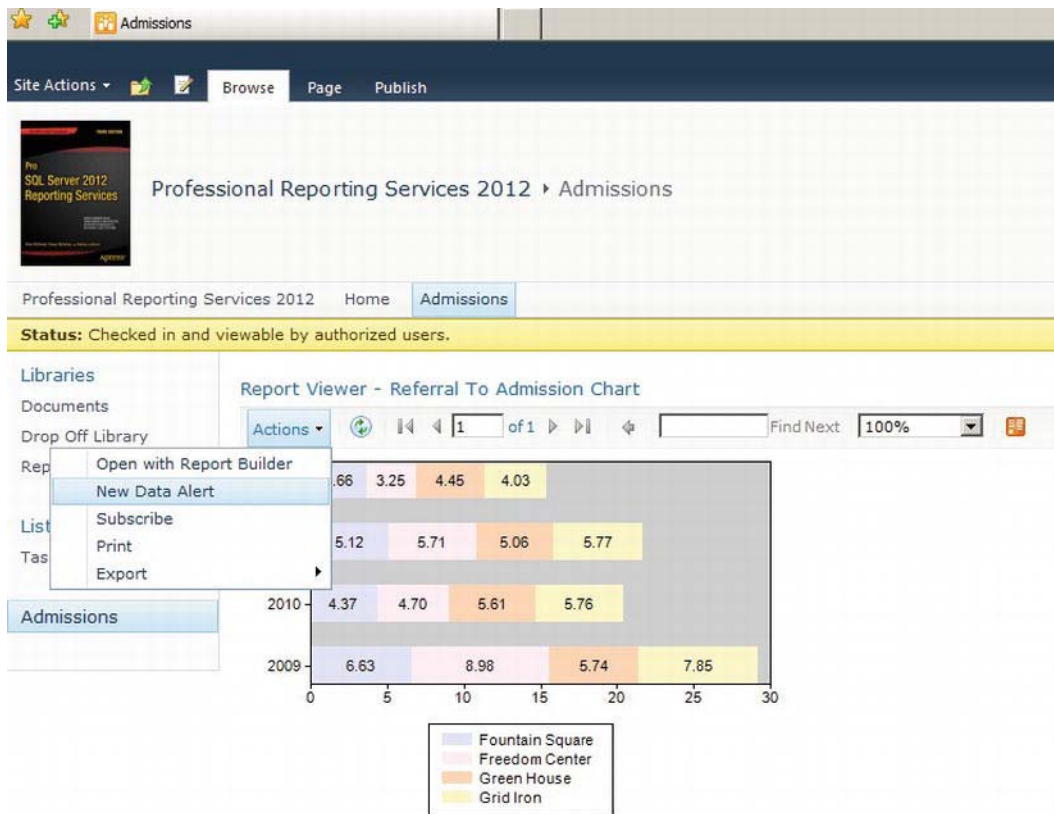


Figure 12-36. Creating a data alert

2. On the New Data Alert screen, configure the data alert by giving the alert a name. In this case, name it DA_Referral to Admission Chart_GT_8.9 as shown in Figure 12-37.
3. Next, click the Add rule... button to create a rule that states when the value of the Category Group1 is greater than 8.9. This will cause the SQL Server Agent to send out an alert when the value is over 8.9.
4. Set the schedule to run once a day. The scheduler is much like the one provided by Report Manager.
5. Enter the recipient to be alerted when the value reaches the desired threshold, the subject, and any description if desired.
6. Click the Save button to have SharePoint create the alert.

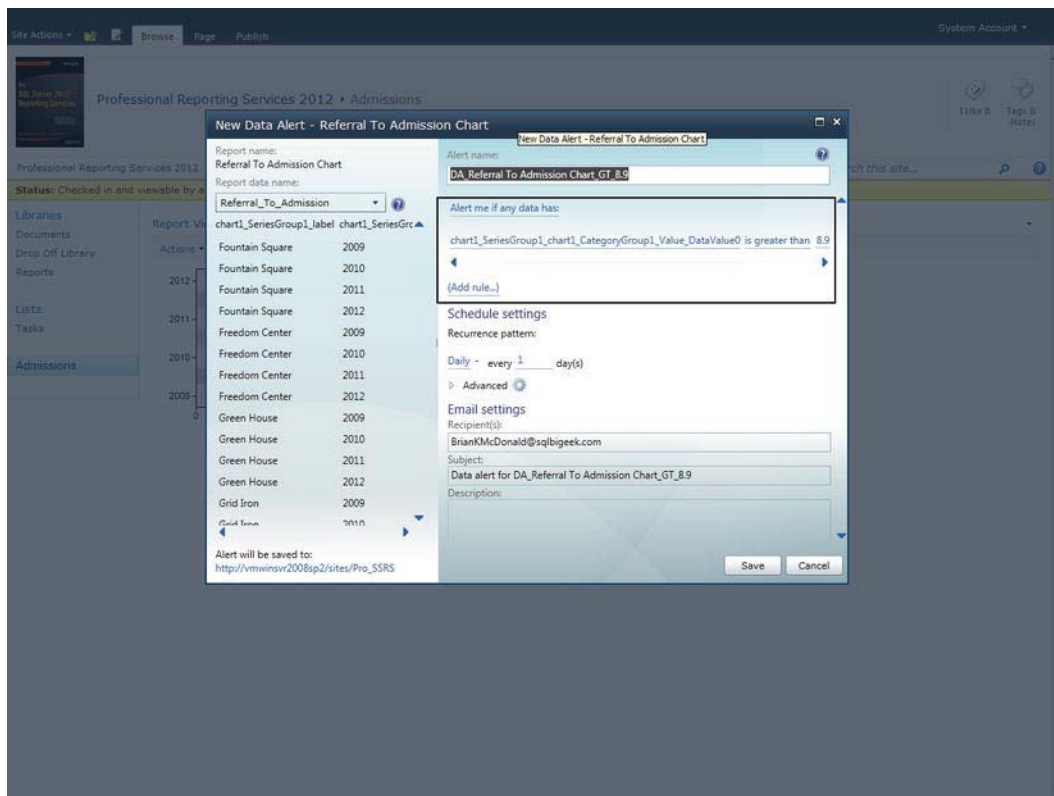


Figure 12-37. *Configuring a data alert*

Now, if the 8.9 threshold value is ever exceeded, you will be sent an email notification that the threshold has been met.

Summary

In this chapter, I showed you how to incorporate SSRS into a BI model, using the business model of a health-care agency deploying a custom BI portal. Understanding how to transform and analyze the data that drives your business will help you make important business decisions. Delivering that data to decision makers is a pivotal link in the BI chain. With SSRS, Microsoft has provided organizations with another tool that can easily tap into and extend the reach of crucial data. SSRS utilizes many types of data and can deliver that data in a variety of formats and mechanisms. Working with other applications and products in the Microsoft BI platform—such as SharePoint, Analysis Services, and Office—SSRS will prove to be an invaluable BI tool now and in the future.

Creating Reports Using Report Builder 1.0, 2.0, and 3.0

When it comes to user-developed reports, we often have to deal with the problem of providing the underlying data in a controlled and secure way, but with enough freedom for that data to be useful for making business decisions. In SQL Server 2005, Reporting Services included the first version of the Report Builder. Report Builder is an application that offloads some of the tedious report design tasks to the report consumers themselves.

In Report Builder 1.0, users with appropriate permissions could create and publish their own reports with one caveat: the data had to be provided using a predesigned report model. This report model created an abstraction layer, if you will, that allowed the users to use the provided attributes as report elements. As with any new advances in software development, we often find some functionality is deprecated. In that regard, SQL Server 2012 is no different. We can no longer create Report models using the 2010 version of BIDS, SSDT, or Visual Studio, but we can still utilize them as a data source within our reports.

With the release of SQL Server 2008 came the Report Builder 2.0 application. With its Ribbon technology, akin to other Microsoft Office products, Report Builder 2.0 delivered a familiar interface and trumped Report Builder 1.0 by allowing users to write queries and use stored procedures to access data directly from the database, without the extra abstraction layer of a report model. This new interface was intuitively easier to use, with capabilities that far exceeded those of Report Builder 1.0.

In April of 2010 came the release of SQL Server 2008 R2 and with it came another version of the Report Builder that offered dataset caching and integration with the enhanced visualization report controls like the Map and Sparklines, as well as Indicators. Since you are now familiar with producing reports using BIDS, let's turn our attention throughout this chapter to creating practical real world reports in Report Builder 1.0, 2.0 and 3.0 applications.

One thing to note before we get started is that, at the time of writing of this book, the vast majority of organizations are still using prior versions of SQL Server. As such, we decided to keep Report Models and Report Builder 1.0 in this version of the book. If SQL Server 2012 is the only version that your organization owns, or if your company is planning to migrate everything to SQL Server 2012, you can skip over the Report Model and Report Builder 1.0 sections. If your company is one of those migrating to 2012, just know that there is currently no path for migrating Report Models to SQL Server 2012. With that caveat behind us, let's get started.

First, we will explore Report Builder 1.0 and its components. As you will see, even though the user who compiles the report needs to do some design work, the administrator still needs to design the report model that provides data to the Report Builder 1.0 client application. We will walk you through the process of using Report Builder 1.0 from two vantage points—developing and publishing a report model based on user input and then creating your own ad hoc reports based on that model. The three steps we will cover are gathering user feedback, creating and publishing the report model, and finally testing the report model by creating reports with Report Builder 1.0:

Getting feedback: Since we have worked alongside developers for many years, we know that one of the biggest problems with delivering a new application is that sometimes users get the application and expect it to do more than it does, or expect it to do something in a different way than delivered. The expectation from a user's perspective is that the system should do exactly what the user needs. But for a developer, if it doesn't do it in version 1, it will have to do it in version 2. You can avoid this conflict by implementing standard design practices, which always take user feedback into consideration up front. This process is especially important when the same application will be delivered to different companies with users from different backgrounds, each with their own interpretations of how a process should work. In the past few decades, a system analyst would take information from groups of users and translate this to a technical design that the developer could slowly digest. Today, most system analysis work is done directly by the developer, who interacts with users during the design phase of the project. Before we cover how to build a report model to be used by report designers, we will provide a sampling of some of the feedback we have received about the kinds of reports users would like to see.

Building the report model: The report model is the heart of Report Builder 1.0. Report models serve as semantic descriptions of the underlying data source. Because the report model can simplify a potentially complex overall database schema by selecting individual tables, fields, and values, it can be likened to a SQL Server view. Indeed, a key component of the report model, as you will see in this chapter, is a data source view. Once DBAs or developers have created a report model that contains the data to meet the requirements of the users, they can publish it to the report server where it can be used by Report Builder 1.0, which is the client-side report designer. All the data provided to Report Builder 1.0 come from published report models. We will show how to create and publish models in this chapter based on user feedback.

Using Report Builder 1.0: Report Builder 1.0 is one of several design applications that can be used to create and publish SSRS reports without requiring a full development environment like Visual Studio or BIDS. Report Builder 1.0 relies on pre-published report models as its data sources. As you will see, the interface for Report Builder 1.0 is intuitive; in fact, it is ideal for users who need to quickly create reports based on predefined report templates. Report Builder 1.0 not only allows users to design reports, but also serves as the publishing tool to deploy the completed report directly to the report server.

When we begin talking about Report Builder 2.0, where users have more control of the data they are accessing, you'll see that much of the same concepts that you learned with respect to Report Builder 1.0 still apply. Developers may find themselves still creating views and stored procedures based on feedback from a user who will ultimately be designing the reports. Likewise, many developers may decide that Report Builder 2.0 has all of the features they need and will select it as their preferred development environment.

Getting User Feedback

At this point in the book, you have built a report, Employee Service Cost, which breaks down the cost of health-care visits at several levels. You designed this report using BIDS, and it contains several grouping levels, interactive sorting, and drill-down functionality. The dataset for this report is a stored procedure

that provides the desired fields from several tables in the Pro_SSRS database. Like the reports you will build with Report Builder 1.0, this stored procedure and report were created based on feedback from users who needed it. One objective when creating the stored procedure was to design it not only to address the requirements for one report, but also to be versatile enough to be reused for other similar reports. We will cover this issue when we show how to build a report model to meet the needs of the users and the report designers who will be creating and publishing the reports with Report Builder 1.0.

In the health-care industry (namely, in post-acute care such as home health and hospice), you have to follow many reporting requirements for state and federal agencies. From cost reports to patient admission and discharge information, different reports are needed by different departments for different reasons. One group of reports that is used by every department is the patient census. This report shows all the patients who are currently admitted to the health-care facility and who are receiving services. The report can be as simple as an alphabetized listing of patient names or as detailed as all the demographic information about each patient, such as where they live and who to contact in case of an emergency.

Since the patient census is the most frequently requested report for customization, we will use it as the example in this chapter for the report model, making sure we provide enough data to meet both current and future needs.

For the patient census report example, imagine that users have requested to see a daily total of patients who have been admitted to the facility. Such patients all have common data associated with them that is important when looking at a census. The information answers these types of questions:

- What is the patient's primary diagnosis?
- What branch office is the patient currently admitted to?
- What is the patient's length of stay in the facility?
- How many active patients are in each branch currently?
- What is the patient's address, phone number, and other key demographic information? (This type of census report, often referred to as a face sheet, shows a lot of information about a patient in a compact location.)
- If the patient was discharged, when did that occur and what was the reason? Is it because they recovered, or did they transfer to a hospital?

■ **Note** The type of information we are discussing in this chapter is considered to be Protected Health Information (PHI) data, which, thanks to HIPAA, needs to be tightly guarded and accessible only to those who are permitted to use it. Chapter 11 covers how to secure access to data delivered through SSRS.

You can answer all of these questions and more with a single query, which can form the foundation of the report model. So, in the Pro_SSRS_2008R2 database, you will join several tables into a single data source view to provide the information needed to deliver the patient census. You have used most of these tables already when developing other reports, but we will review the database schema of the associated tables before showing how to build the report model. These are the tables you will use in this chapter's example:

- Admissions

- Branch
- Diag
- Discipline
- DocumentImage
- PatEMRDoc
- Employee
- PatDiag
- Dischg
- Patient

Each table contains specific information that is relevant to the individual patient's admission history, and together they will provide enough data to create many kinds of census reports for many departments within the health-care agencies. You know the kinds of questions you will need to answer, and the tables have been designed to deliver that information based on direct feedback and requests for users. Now, let's move on to designing the query and building the report model.

Introducing the Report Model

Just because you don't understand the intricacies of putting all the parts of a car engine together does not mean you can't drive, right? The same can be said for report designers who will design and publish reports using the ad hoc Report Builder 1.0. In this case, you have little need to understand the pieces and parts of the database structure in order to build reports. It is unavoidable, though, that *someone* must understand the database schema, just as mechanics must understand brake pads and fan belts. Building a versatile report model, which the report designer will use to create reports, will most likely fall under the authority of the DBA, data analyst, or SQL developer, who will have an intimate understanding of the data sources. In this section, we will show how to use the feedback gathered from the report consumers to develop a report model for the patient census reports that will serve to make the data easily accessible with logical and intuitive field names.

You will perform the following procedures to create the report model for the Patient Census report model:

1. Create a report model project using BIDS/SSDT.
2. Create a data source.
3. Create a data source view.
4. Create the Patient Census report model based on the data source view
5. Publish the Patient Census report model to the SSRS server.

The process of building a report model consists of many individual steps, many of which are controlled entirely by stepping through available wizards. A report model can be complex or simple, based on your individual needs. For the sake of brevity, we will keep the sample model as simple as possible; you will be able to answer the feedback questions for the patient census, and you will be able to create and publish your own report model without any issues.

Create a Report Model using BIDS 2008 R2

In this section, you will create a report model project using BIDS 2008 R2. A completed version of this project is included in the Source Code/Download area for the book on the Apress Web site (www.apress.com). When you open the solution, you will notice an already created report model called Patient Census Book. If you would like to publish the Patient Census Book report model as it is in the project to your SSRS server and begin working with the client-side Report Builder 1.0 application, just be sure to set the data source settings and deploy appropriately. In that case, just skip to the “Creating Reports with Report Builder 1.0” section. To follow the steps to create each piece of a report model using Business Intelligence Development Studio for SQL Server 2008 R2, from adding data sources and data source views to publishing the completed model, read on.

Open up BIDS on a machine that has BIDS installed for a SQL Server 2008 R2 installation. You will find BIDS by clicking on the Windows Start button, All Programs, SQL Server 2008 R2, and then SQL Server Business Intelligence Development Studio. Once BIDS is started, click on Project... next to the “Create:” label. As you can see in Figure 13-1, you can use several available project templates. Choose the Report Model Project template, name it Patient Census, add the appropriate location (in this case C:\Pro_SSRS Project_2008R2\), and click OK.

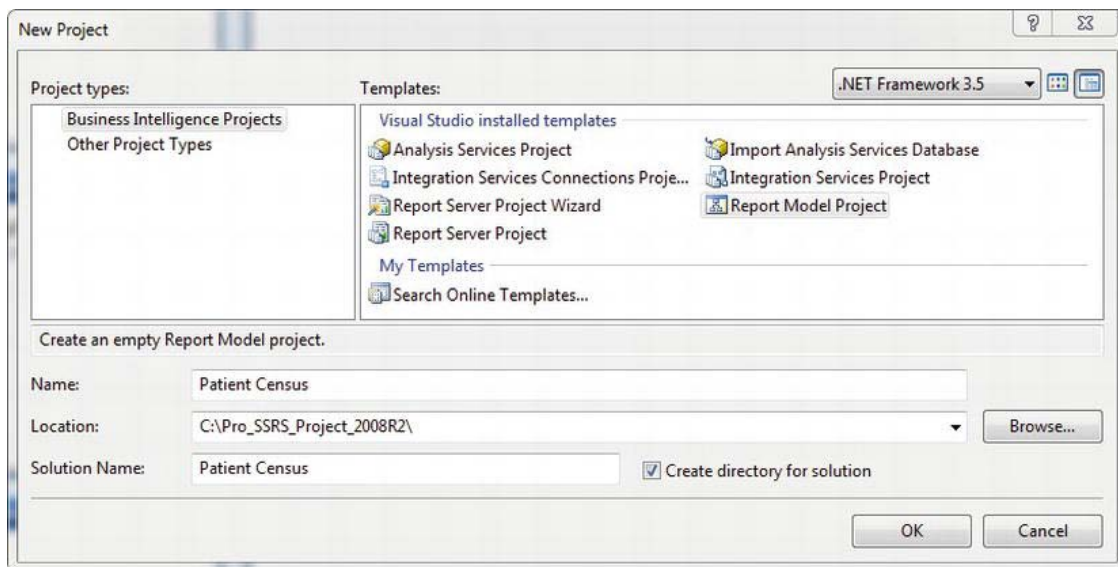


Figure 13-1. Creating a new report model project

A report model project has three main components: data sources, data source views, and report models, each one relying on the previous one. In other words, a data source view requires a data source, just as a report model requires a data source view. If you have successfully created the Patient Census report model project, you will see these three components, as shown in Figure 13-2.

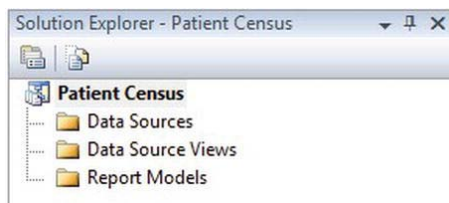


Figure 13-2. Viewing the new Patient Census report model

Adding a Data Source

Logically, the first step is to create a data source. Before we can create a data source connecting to our SQL Server 2008 R2 machine, you will need to restore the Pro_SSRS_2008R2 database, as described in the ReadMe.txt file available in the Source Code/Download area for the book on the Apress Web site (www.apress.com). After you have the Pro_SSRS_2008R2 database set up, we need to create a data source for our model. Right-click the Data Sources folder in the Solution Explorer and select New Data Source under the Add menu. This brings up the Data Source Wizard to create a data source for a report model. Figure 13-3 shows the first page of the Data Source Wizard. Fortunately, you can select to not display the first page every time you open the wizard by checking Don't Show This Page Again.

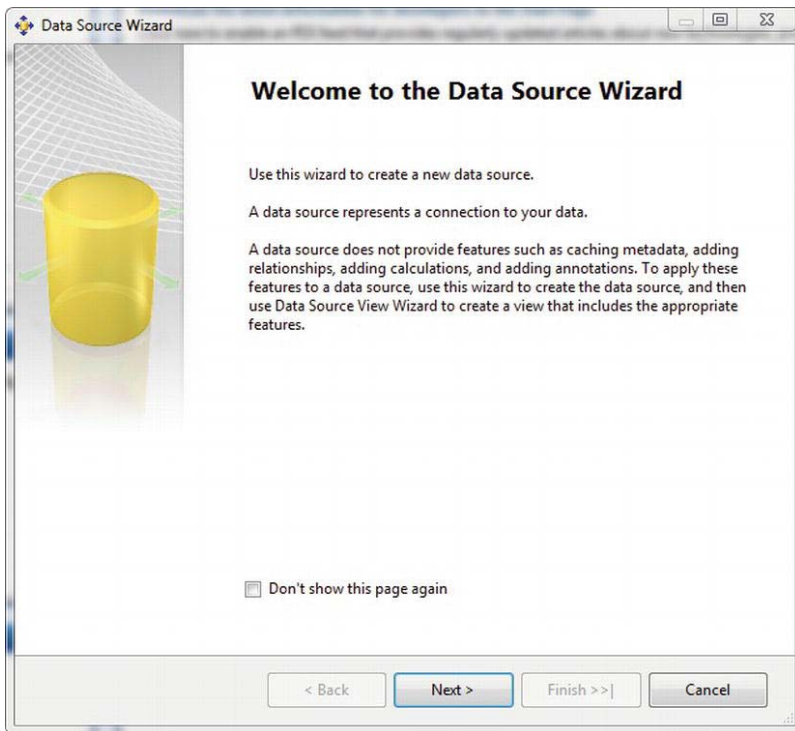


Figure 13-3. Using the Data Source Wizard

You have two options when creating a data source through the wizard, as shown in Figure 13-4. You can either create a new one based on an existing or a new connection, or you can create a data source based on another object such as a data source already defined in another project. For this example, choose to create a new data source based on a new connection, and click New to open the Connection Manager.

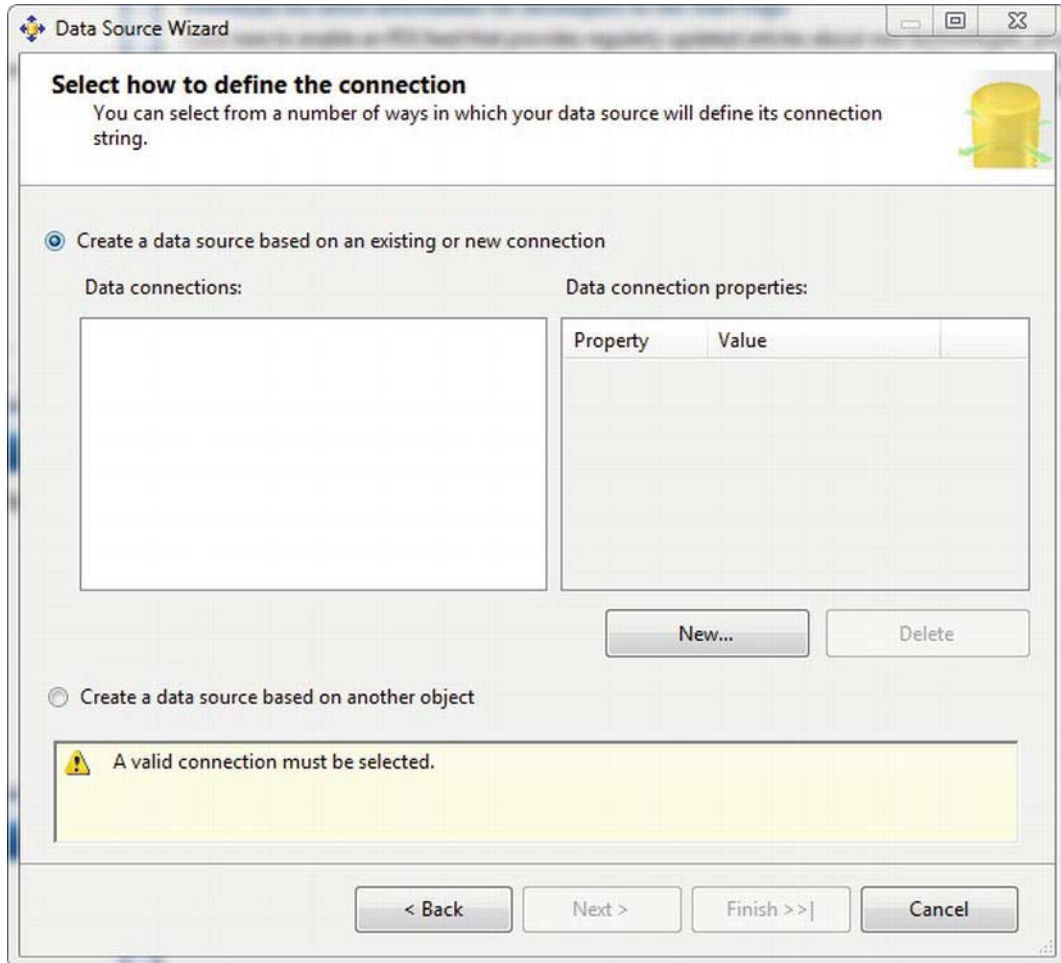


Figure 13-4. Defining a connection

In the Connection Manager, select localhost as the server name, leave the Use Windows Authentication button checked, and change the database name to Pro_SSRS_2008R2, as shown in Figure 13-5. Click OK to save the settings and return to the Data Source Wizard. Click Next or Finish to proceed to the final screen of the wizard, leave the default data source name of Pro SSRS 2008R2, and click Finish to create the data source.

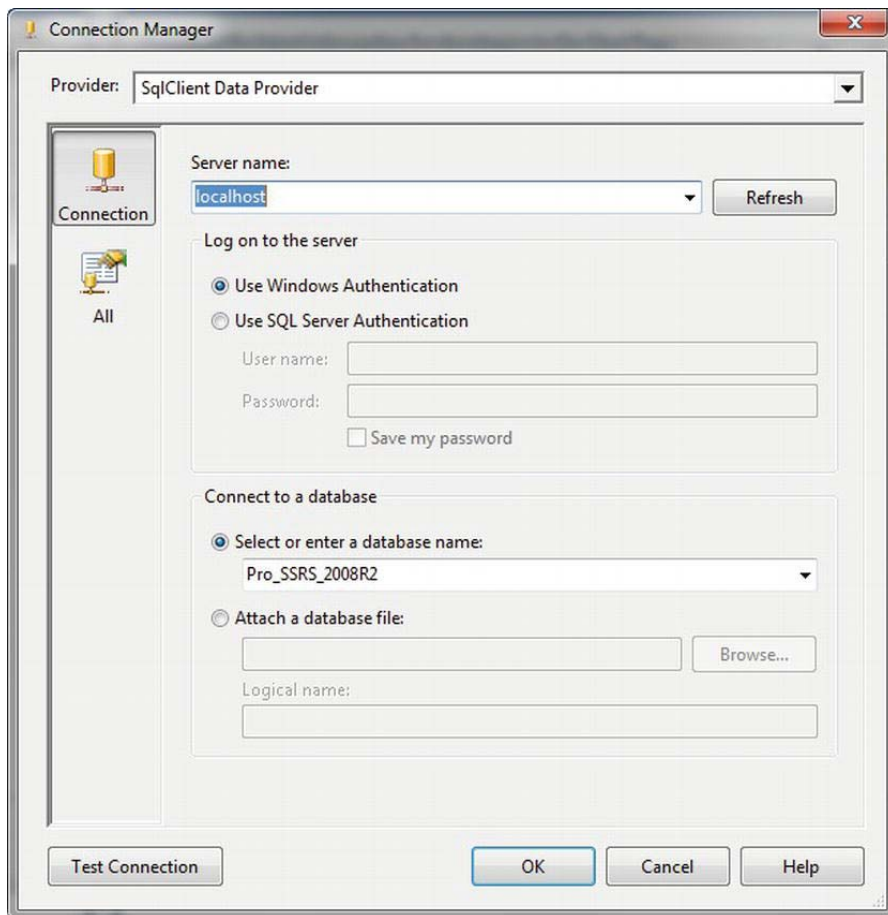


Figure 13-5. Setting the server name and database

■ **Note** The Pro_SSRS_2008R2 database is a version of the Pro_SSRS database that we have been using throughout the book. However, since Report Models have been deprecated in SQL Server 2012, we created a 2008 R2 version of the database with limited objects. The Pro_SSRS_2008R2 database and script can be found as part of the Source Code/Download area for the book on the Apress Web site (www.apress.com). For more details on this database, be sure to read the ReadMe.txt file found in the Source Code/Download file.

Creating a Data Source View

You create data source views for a report model project from relational data sources, where you have a list of tables that can be joined to form the basis of the data source view. In the Pro_SSRS_2008R2 database, you know many related tables are joined by column ID fields; for example, the Admissions table joins to the Patient table via the PatID field. Fields in both tables may be relevant to a user designing a report; however, superfluous data may just confuse that user. Part of your job as the report modeler is to remove the complexity of the schema but also provide a model that contains intuitive or “friendly” names and provides only the data that will be useful. Why should you use Dscr for a diagnosis code, when you really should just make it Diagnosis Name, for example? Since it is also possible to add objects to a data source view that are derived tables made from custom queries, you will have more control over creating a custom data source view and eliminate much of the work of removing the extraneous fields included in all the tables you are using. This will become more evident to you as you step through the process. For now, we will show how to perform the following logical steps to keep the data source view and subsequent model simple but effective:

- Building a query instead of many multiple tables for the data source view
- Selecting only relevant and potentially useful fields for the report designers, based on their requests
- Using friendly names for data source view fields and values

The first step in creating the view, like with the data source, is to open the wizard by right-clicking the Data Source Views folder and selecting Add New Data Source. Clicking Next to proceed to the next screen in the wizard will bring you to a screen depicted in Figure 13-6.

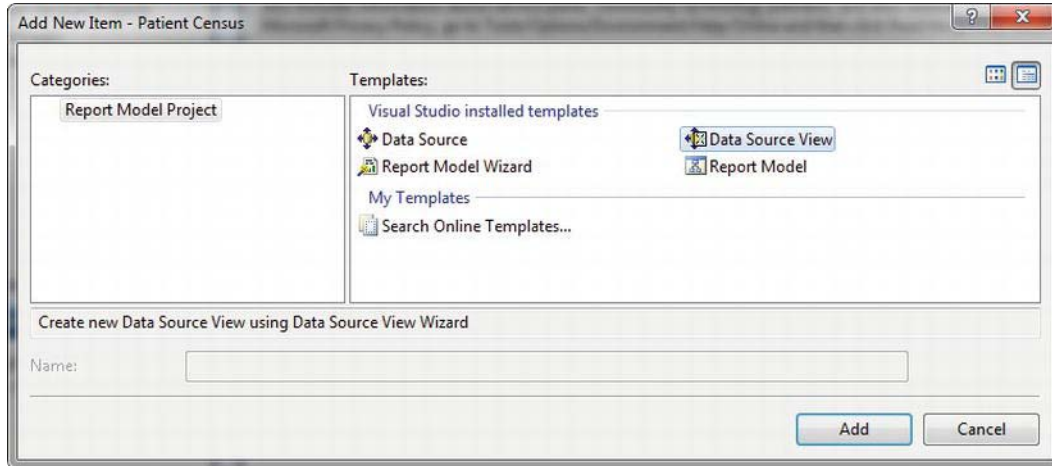


Figure 13-6. Adding a data source view to the project

On the first page of the wizard (not the welcome page, on which you should immediately check Don't Show This Page Again), select the Pro_SSRS_2008R2 data source you created earlier, and then click Next to move to the Select Tables and Views page. As you can see in Figure 13-7, you can choose from many tables.

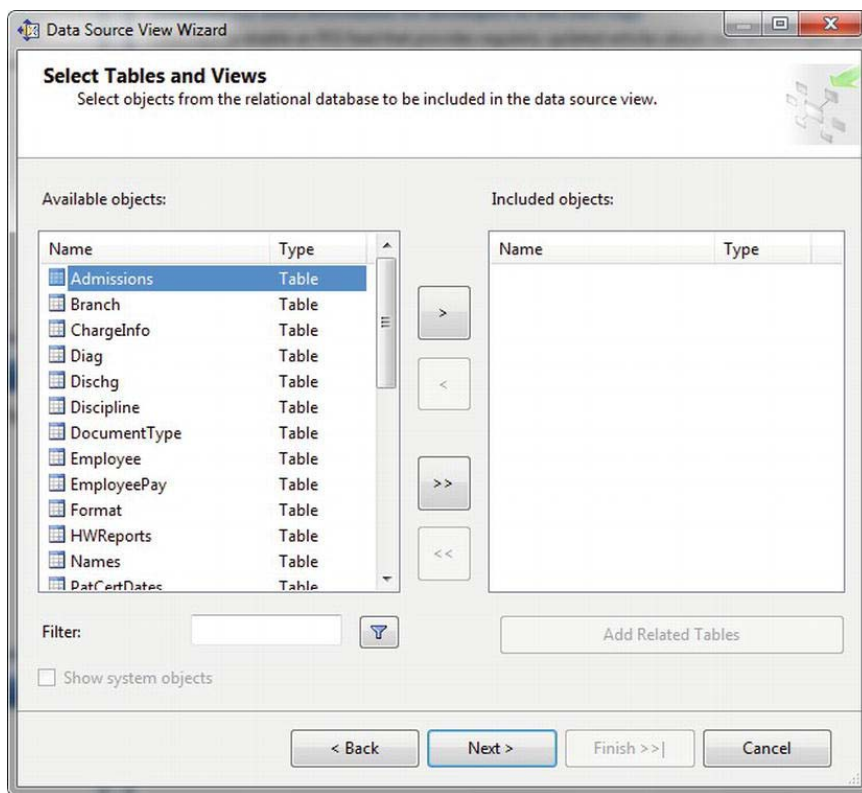


Figure 13-7. Related tables in the Pro_SSRS_2008R2 data source

You could add all the tables from the list of required tables for the patient census reports you are going to create; however, knowing that you will have too many unnecessary fields to work with, simply select Next and then Finish to close the wizard without including any objects. This will create the Pro SSRS 2008R2 data source view with no tables defined. Why, you may ask, did you not select any relational objects, since this seems like a crazy way to start? The reason is that you will add your own single object, called a *named query*, which is like a view in a SQL Server database. It is an object derived from a query of other objects. Assuming you had chosen one or more tables, you would have graphically defined the relationships for the tables based on primary keys, if they existed, or by using related column ID values and these tables would become the source for the report model. The end result is the same, whether you use real table objects with relationships or you use named query tables that contain only the fields you want. The difference is that this way you gain the benefit of one simple and concise table, instead of using many tables with the potential downside of adding more data than are needed for the model.

Once you have created the data source view, which defaults to a name of Pro SSRS 2008R2 based on the data source, you can double-click to open it in the design environment. It will appear empty, as it should. You can add the source query, or more accurately, the named query, by clicking the New Named Query button on the toolbar, as shown in Figure 13-8.

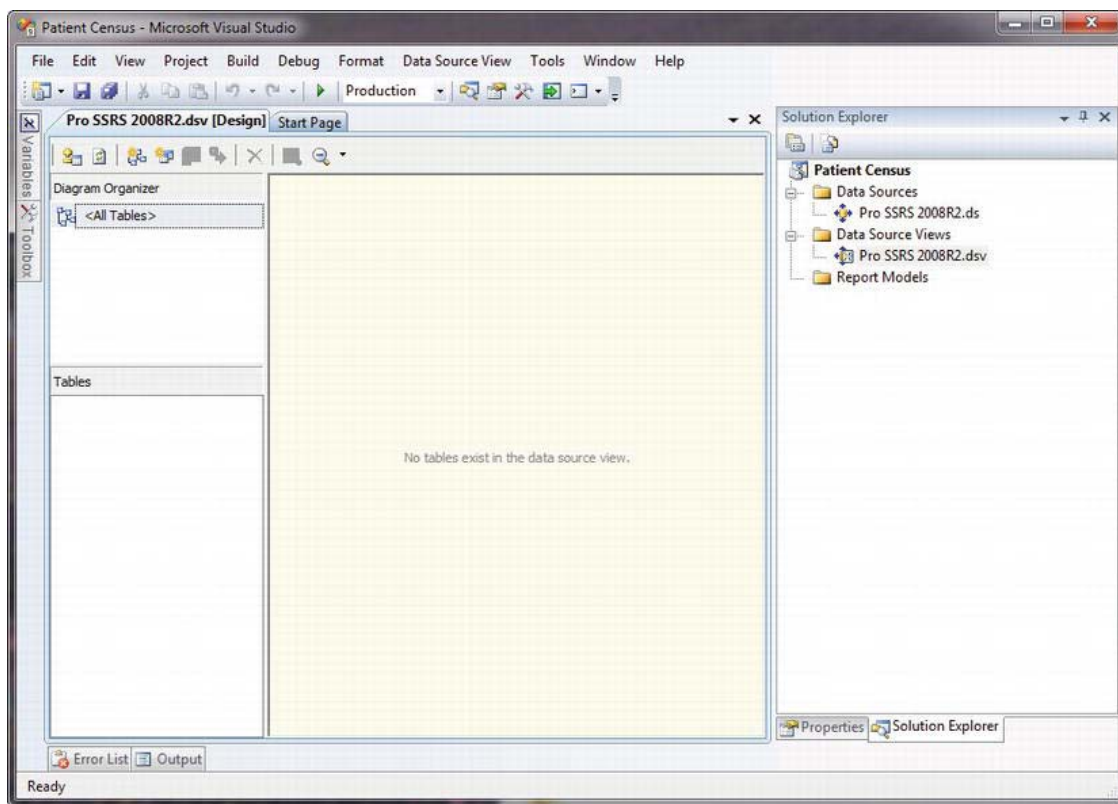


Figure 13-8. Pro SSRS data source view

In this case, you have already designed the query; however, Report Builder 1.0 includes its own graphical query builder, so it is possible to develop the query on the fly, so to speak. The graphical query designer is standard, with standard diagram, grid, SQL, and result panes. Clicking the New Named Query button opens the Create Named Query window. In the SQL pane, we can paste the pre-existing Patient Census query from Listing 13-1, and click in the blank diagram pane to automatically load the graphical representation of the query. If you need to provide aliases for any fields, such as distinguishing the common LastName fields from the employee and patient table, you can do that in the grid pane, as shown in Figure 13-9, which also shows the query and all the joined tables. Alias the lastname field as Pat_LastName for patient and Emp_LastName for employee. Otherwise, if there were common references to lastname, the default alias would have been ExprX, where X is the number of common references. You can do the same for the ubiquitous DSCR, used for descriptions, as in the Diag and PatEMR tables. We have also created an alias for each of the tables. This cuts down on the typing and makes the SQL easier to follow. You can alias the table names in the SQL section of the code or by Right-Clicking in the header section of each table and selecting Properties. You can enter the alias in the Alias section of the properties window.

■ **Note** To make it easier on you with less typing, we have included this query in the source code for this book in a file called Patient Census Query – Chapter 13.sql. Of course if you really like typing, though, feel free to write the SQL statement provided in Listing 13-1.

Listing 13-1. Patient Census Query

```
SELECT
    A.PatProgramID
    , E.EmployeeID
    , E.LastName AS Emp_LastName
    , E.FirstName AS Emp_Firstname
    , D2.Dscr AS Discipline
    , B.BranchName
    , P.PatID
    , P.LastName AS Pat_LastName
    , P.FirstName AS Pat_FirstName
    , D.DiagID
    , D.Dscr AS Diagnosis
    , PD.DiagOnset
    , PD.DiagOrder
    , A.StartOfCare
    , A.DischargeDate
    , P.MI
    , P.Address1
    , P.Address2
    , P.City
    , P.HomePhone
    , P.Zip
    , P.State
    , P.WorkPhone
    , P.DOB
    , P.SSN
    , P.Sex
    , P.RaceID
    , P.MaritalStatusID
    , EMR.DateEntered
    , EMR.Dscr AS EMR_Document
    , DS.Dscr AS [Discharge Reason]
    , DATEDIFF(dd, A.StartOfCare, A.DischargeDate) + 1 AS [Length of Stay]
FROM
    Admissions AS A
    INNER JOIN Patient AS P ON A.PatID = P.PatID
    INNER JOIN Branch AS B ON B.BranchID = P.OrigBranchID
    LEFT OUTER JOIN PatDiag AS PD ON A.PatProgramID = PD.PatProgramID
    INNER JOIN Diag AS D ON PD.DiagTblID = D.DiagTblID
    LEFT OUTER JOIN Employee AS E ON A.EmployeeTblID = E.EmployeeTblID
    LEFT OUTER JOIN Discipline AS D2 ON D2.DisciplineTblID = E.DisciplineTblID
```

```

LEFT OUTER JOIN PatEMRDoc AS EMR ON A.PatProgramID = EMR.PatProgramID
LEFT OUTER JOIN DocumentImage AS DI ON DI.DocumentImageID = EMR.DocumentImageID
LEFT OUTER JOIN Dischg AS DS ON A.DischargeTblID = DS.DischgTblID
WHERE
(PD.DiagOrder = 1)

```

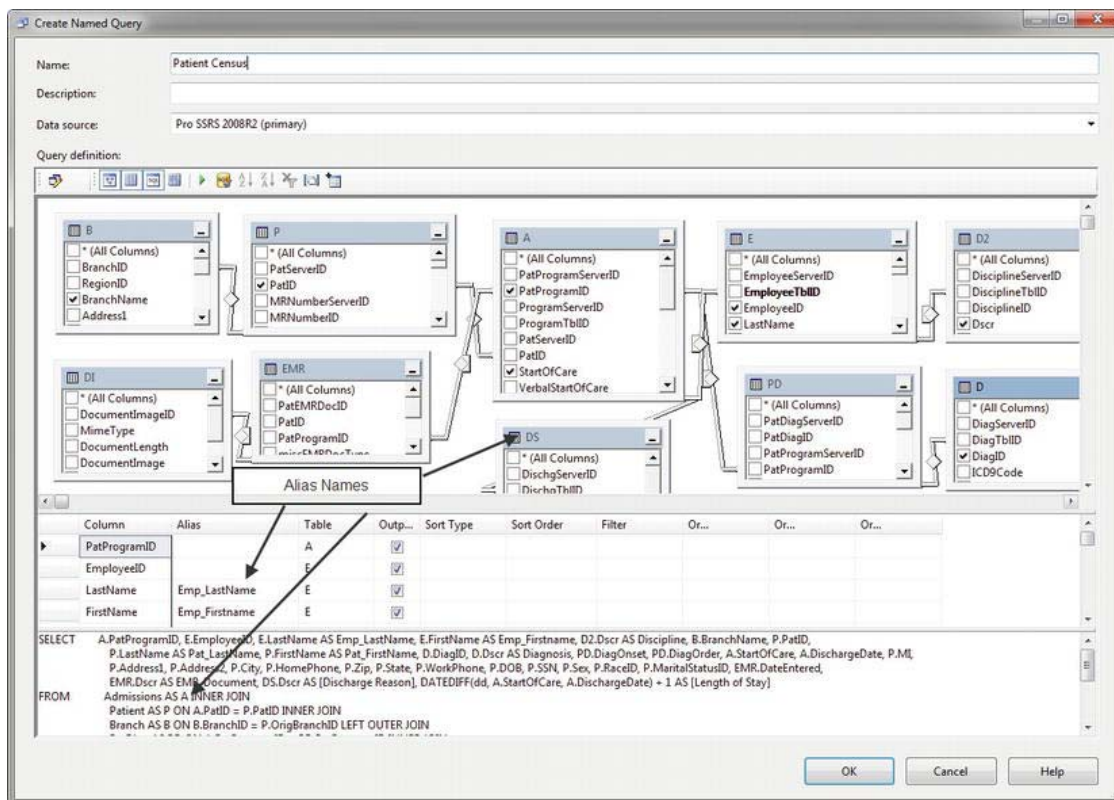


Figure 13-9. Named query displayed graphically with alias names

Finally, name the query Patient Census. When you click OK to complete the named query, your simple data source view consists of one object, Patient Census that contains the desired fields for the model. You can see in Figure 13-10 that 32 fields will be supplied to the model.

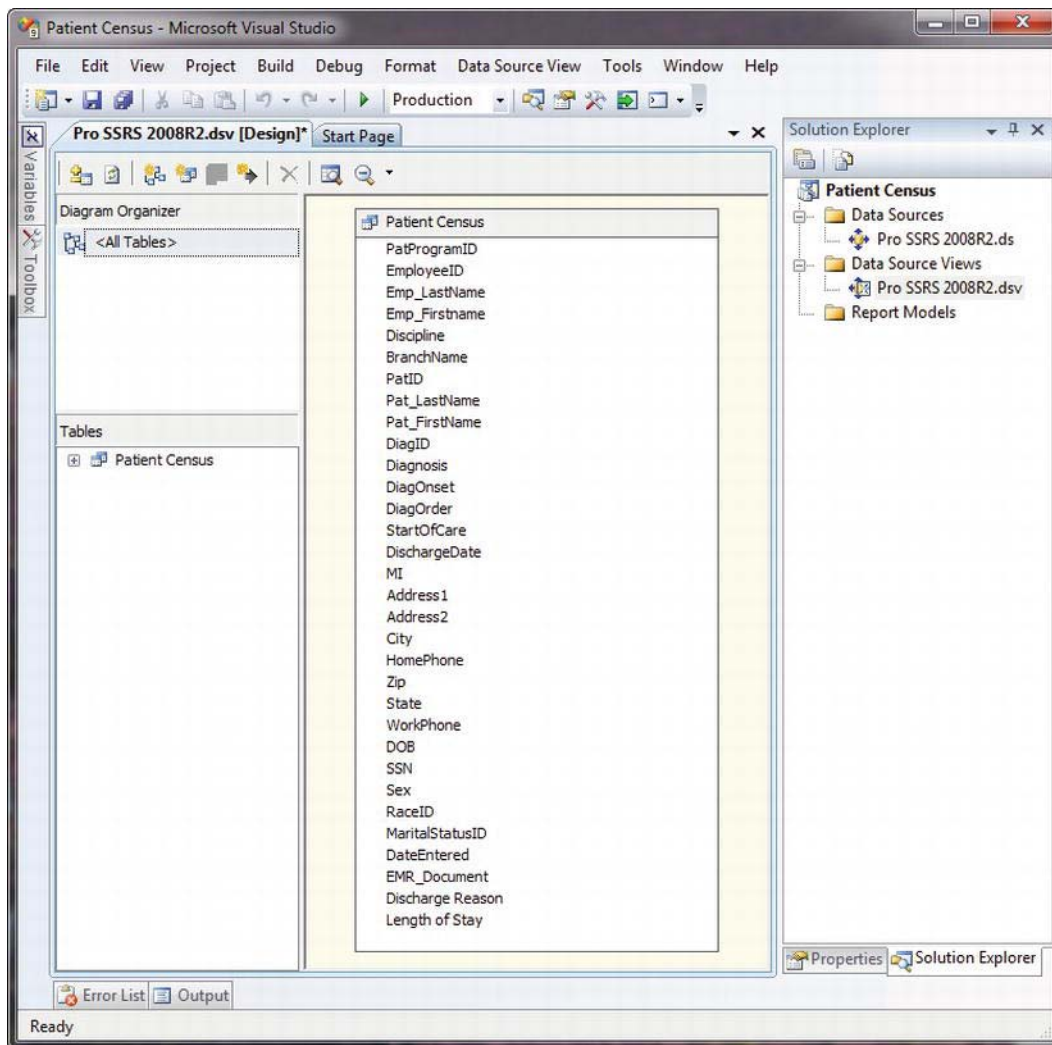


Figure 13-10. Data source view with one object

The next step in the process is to add a logical primary key to the Patient Census named query because the report modeler requires a primary key when it generates the model. Because the query is driven from the Admissions table, which contains the PatProgramID field that you know will contain unique values, you can use this field as the logical primary key for the Patient Census query. The PatProgramID field, in the example database, controls the admission and discharge records for a patient over time. It is typical that the same patient will be admitted to two separate service lines, referred to as *programs*; hence, you have PatProgramID. Without the PatProgramID field, the records would not be unique with the patient's identification number, PatID, alone.

To set the logical primary key for the Patient Census object to be PatProgramID, simply right-click the PatProgramID field either in the Tables list or in the Patient Census diagram, and select Set Logical Primary Key, as shown in Figure 13-11. When you do this, you will see an icon that looks like a key appear next to the PatProgramID field.

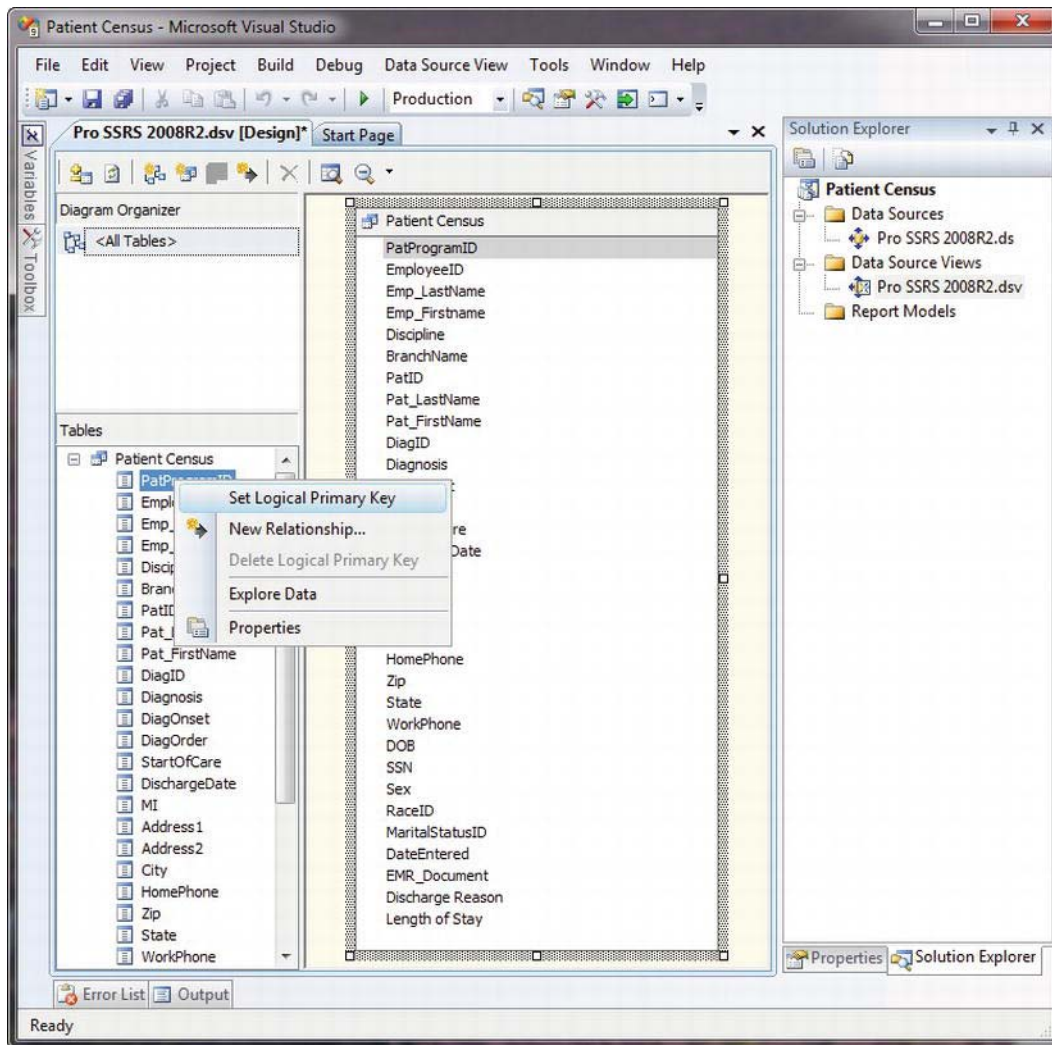


Figure 13-11. Setting the logical primary key

At this point, you can save the data source view and then create the report model. Before you do, however, let's take a look at a really great feature in the design environment, which is the ability to explore the data. Right-click anywhere in the Patient Census object—on EmployeeID, for example—and select Explore Data. As you can see, many records are returned from the entire Patient Census object, which you will use for the report you create later. However, also notice the three additional tabs beside

Table, as shown in Figure 13-12. The Pivot Table, Chart, and Pivot Chart tabs suddenly appear, like finding a really valuable trumpet shell unexpectedly on the beach. (Not that this will help you greatly when developing the report model, but we just thought it was worth showing.)

Now that you have determined that the Patient Census data source view is returning data as you expected, you will save it and move on to the real heart of Report Builder 1.0, the report model.

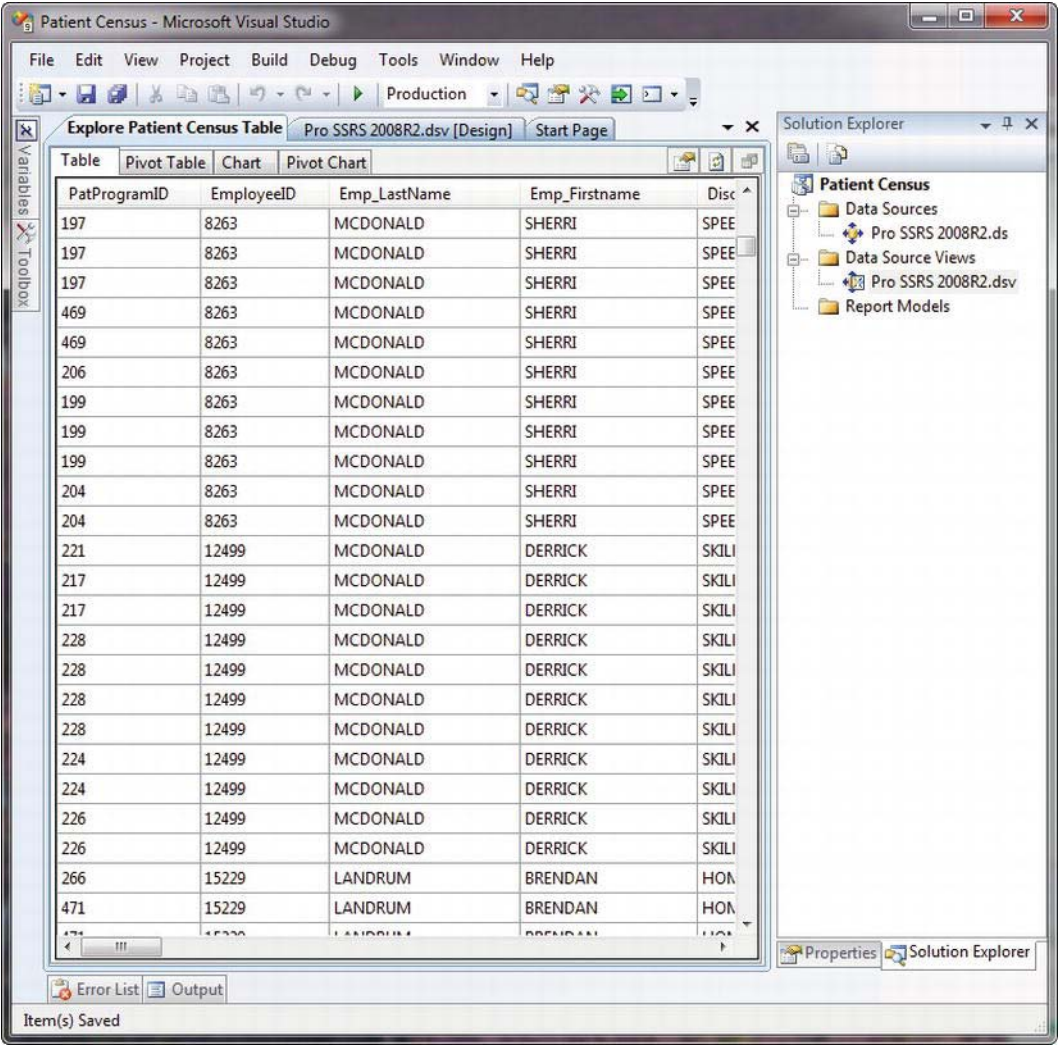


Figure 13-12. Additional tabs when exploring data

Creating a Report Model

The report model will add a layer of separation between report designers and the underlying data source, simplifying the process of designing reports by allowing them to focus on report design and not query design. Now that you have the data source view, which is a requirement for building a report model, you will learn how to create the model. The process is straightforward, thanks to auto-generation methods that analyze the data source view in two passes and produce the end result—a publishable report model in which the constituent elements would have otherwise taken much time and effort to produce, as you will see. The goal in this section is to walk you through the wizard so you can pay attention to the generation methods and finally deploy your completed model to the SSRS server.

You will again utilize the right-click procedure to initiate the Report Model Wizard from the Report Models folder in the Solution Explorer. After navigating past the first welcome page to the Select Data Source View page, you are presented with the Pro SSRS 2008R2 data source view selection, as shown in Figure 13-13.

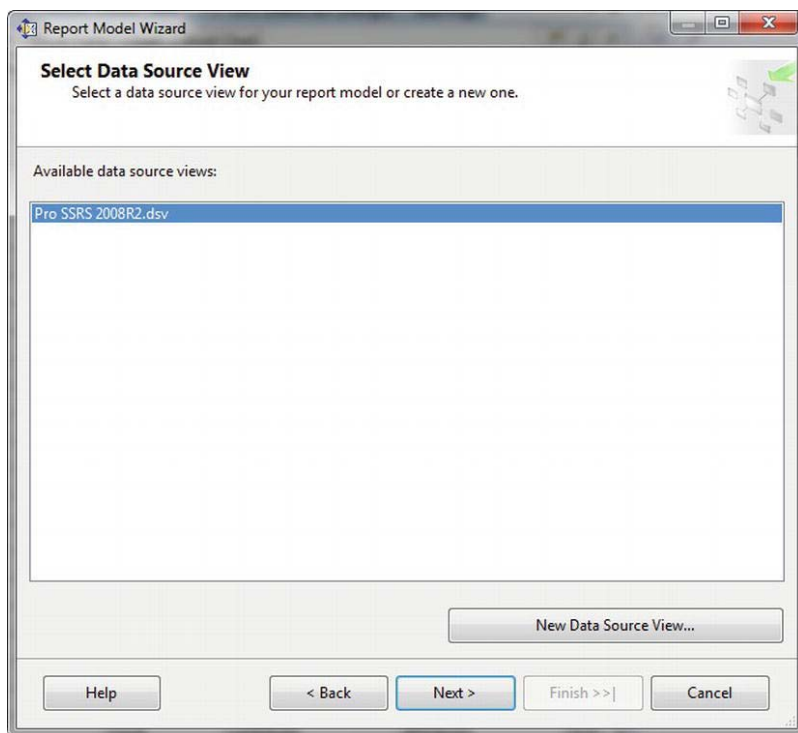


Figure 13-13. *Selecting a data source view*

After selecting Pro SSRS, click Next, where you can now select the report model generation rules. You will see a set of rules that will be performed on the data source view. Several rules exist, and most of them are selected by default. Many of the rules are self-explanatory; however, two are not selected, by default: Create Entities for Nonempty Tables and Create Attributes for Nonempty Columns. These two rules instruct the generation process to create entities and attributes only for tables that contain data.

Entities are the report model’s table equivalent, and attributes are conceptually the fields or columns defined within the entities.

The following are all the rules for the first pass of generating the model and are selectable in the Report Model Wizard:

- Create Entities for All Tables
- Create Entities for Non-empty Tables
- Create Count Aggregates
- Create Attributes
- Create Attributes for Non-empty Columns
- Create Attributes for Auto-increment Columns
- Create Date Variations
- Create Numeric Aggregates
- Create Date Aggregates
- Create Time Aggregates
- Create Roles

This pass processes all rules up to Create Roles. You will also see a helpful description for each rule, as shown in Figure 13-14.

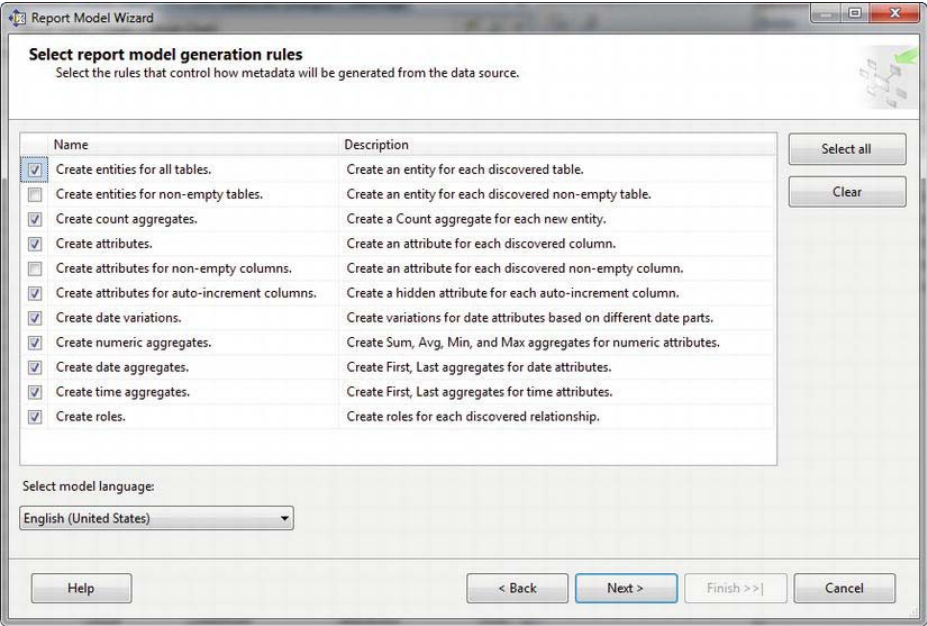


Figure 13-14. Model generation rules with descriptions

For this example, leave the default selections, as these will give you features such as automatic aggregations for any numbers and dates and also create date variations so that for every Date Time data type in the data source view, you will have Day, Month, and Year created. You can click Next and choose to update statistics before generating, as shown in Figure 13-15. This is important if any data have changed significantly in the source tables.

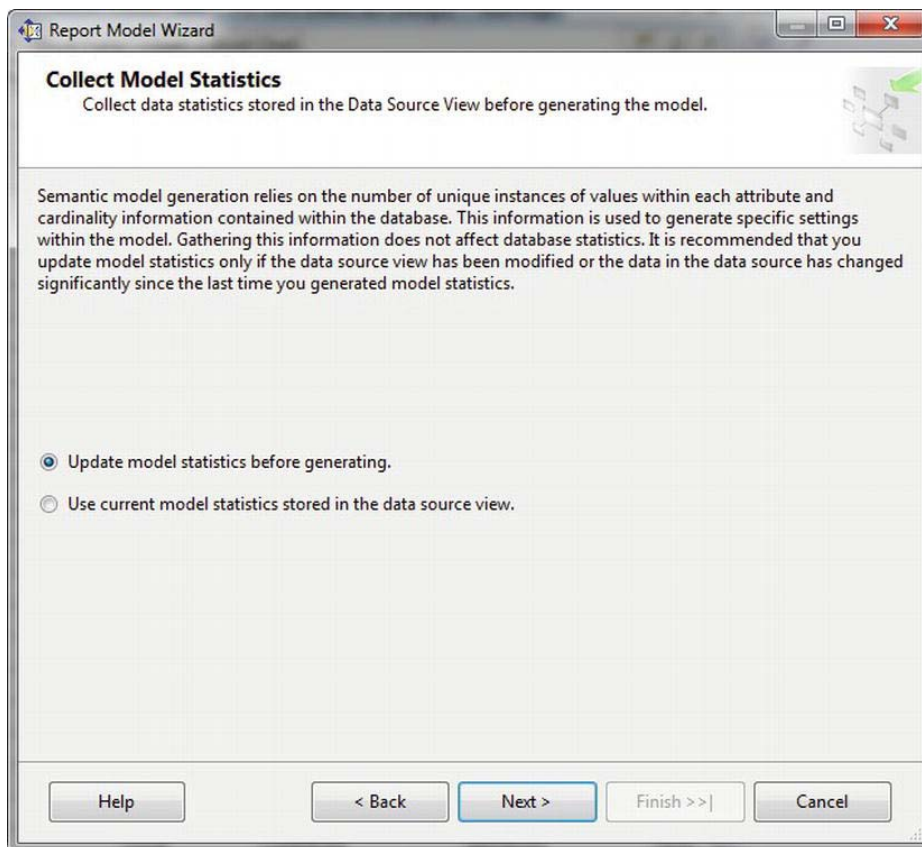


Figure 13-15. Update Statistics page

On the next page in the wizard, name your model Patient Census and select Run to begin the rules to generate the model. Figure 13-16 shows the output after clicking Run, and the process has completed successfully. You can see the details at each rule level of exactly what the rule did based on the data in the data source view. Notice that for Create Date Variations, for example, the rule discovered and created dates for DiagOnset, which is the date on which the patient was diagnosed with a disease. Other attributes that were created were Sum, Avg, Min, and Max for the Length of Stay column. The length of stay, usually an average, is the amount of time between the patient being admitted and being discharged from a long-term health-care facility. The DateDiff function calculates the length of stay.

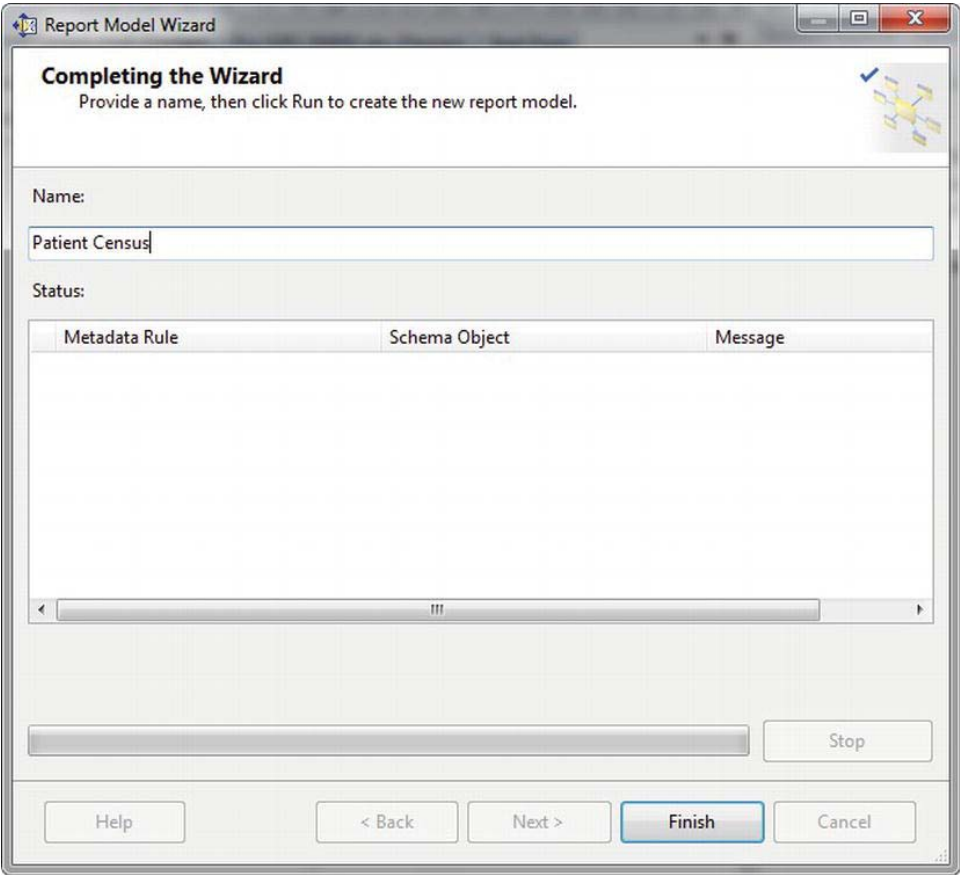


Figure 13-16. Rules completed successfully

After you click Finish in the Report Model Wizard, you can see that the Patient Census model has been created with the expected attributes. At this point, you can deploy the model directly to the report server by right-clicking the model in the Solution Explorer and selecting Deploy. You can control where the model will be deployed in the project's properties pages, as shown in Figure 13-17, by clicking Project Properties under the Project on the toolbar. In this case, the project will be deployed to the Models folder on the report server, which has a target server URL of `http://localhost/ReportServer`. The data sources for the model will be published in the Data Sources folder.

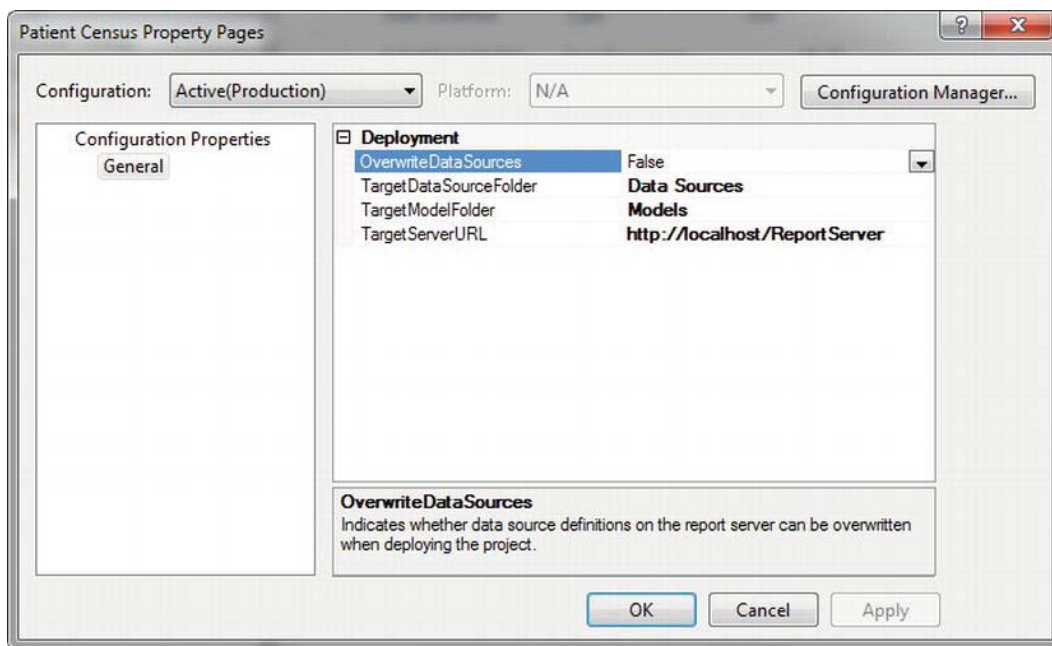


Figure 13-17. Project properties for the report model

■ **Note** We should mention that report models are created with Semantic Model Definition Language (SDML), which is an XML-style grammar similar to RDL. Where RDL defines reports and report properties, SMDL defines model objects, such as the entities and attributes that you are creating here.

After you deploy the model, you can see in Report Manager the two folders that were created, Data Sources and Models. The Data Sources folder contains the Pro_SSRS_2008R2 data source, and the Models folder houses the Patient Census report model. Figure 13-18 shows the properties of the Patient Census report model. Just as with a standard data source, it is possible to view which reports have been created using the model.

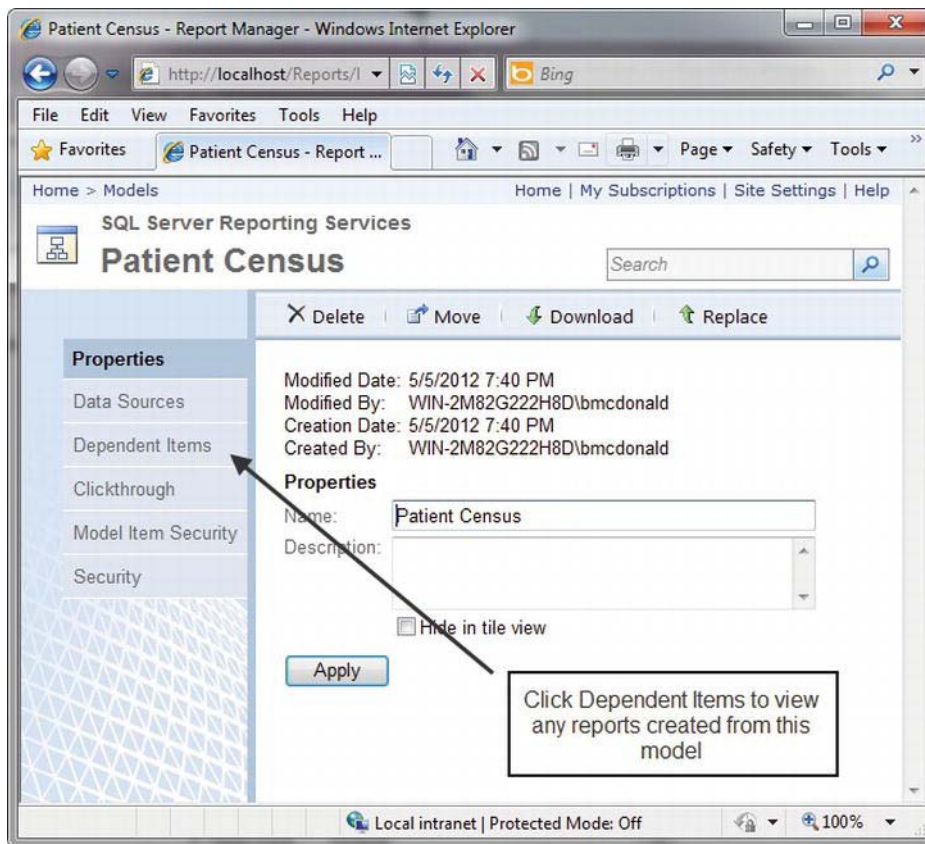


Figure 13-18. Patient Census report model properties

Before you begin creating reports with Report Builder 1.0, it is important to note that users who will need permissions to use Report Builder 1.0 must have item-level role assignments for various tasks associated with creating reports with Report Builder 1.0. Reporting Services includes a Report Builder 1.0 role for this purpose, as shown in Figure 13-19. Users assigned to the Report Builder 1.0 role have all the required permissions to access the report models, folders, and published reports. To publish reports, however, they will need to have the ability to manage report content before they can successfully publish their reports to the report server. The Content Manager role has all the required permissions for full report creation and publishing capabilities.

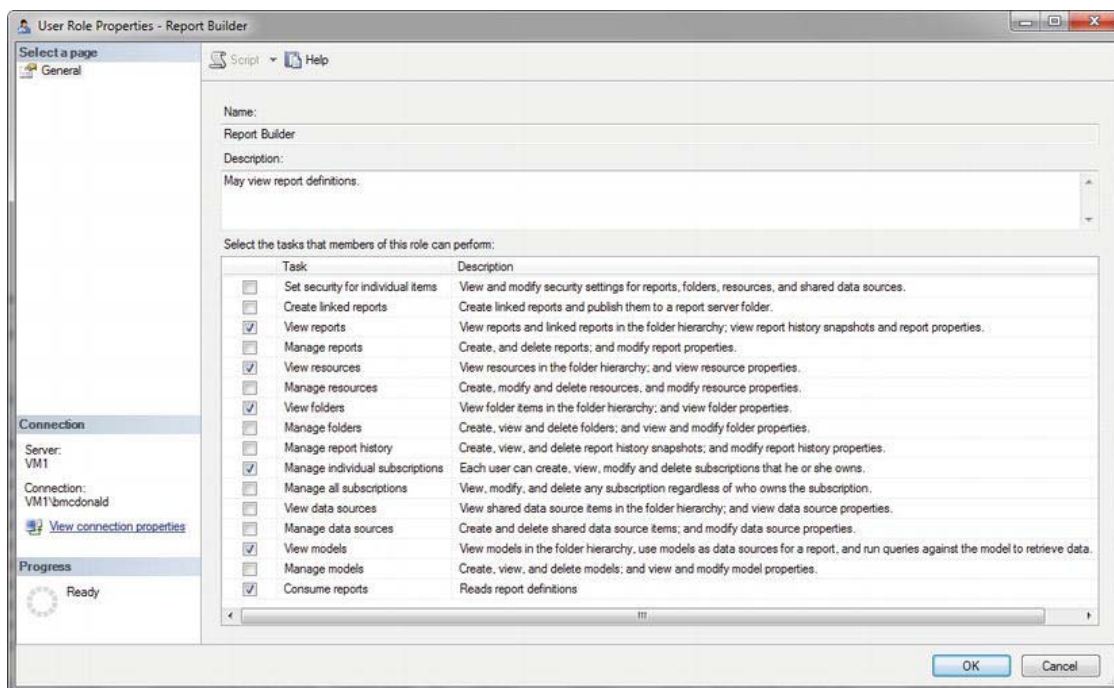


Figure 13-19. SSRS Report Builder role

■ **Note** To manage roles in SQL Server Reporting Services 2008 R2, we have to connect to the Reporting Services instance by running SQL Server Management Studio as an administrator. For more details on connecting to the Reporting Services instance and roles, please see the “Introducing SSRS Roles” section in Chapter 11.

Creating Reports with Report Builder 1.0

To this point you have created all the requisite pieces for using Report Builder 1.0—you have created the report model Patient Census and deployed it to the report server. It is now waiting to be used as a source for the front-end Report Builder 1.0 application. Report Builder 1.0 provides some of the same functionality of a report development environment, such as BIDS or Visual Studio, including the ability to drag and drop data elements to the design area with Matrix, Table, and Chart data regions as well as the ability to deploy finished reports to the report server. However, before you dive in, we should state that Report Builder 1.0 serves the purpose of allowing end users to design their own reports. It was designed to be intuitive and friendly, and because it is not a full-featured IDE, it may at first seem limited.

In this section, you will explore Report Builder 1.0 and uncover the features that are available to the report designer such as adding functions, similar to adding expressions in BIDS, and providing filtering,

grouping, and sorting capabilities. The goal in this section is to show you how to create and deploy, step by step, the requested census reports from the Patient Census model created earlier in this chapter.

When Reporting Services is installed for SQL Server 2008 R2, the default Report Builder application is 3.0. If you are following along or you want to create reports using Report Builder 1.0, you will either need to navigate directly to the Report Builder URL or change the default Custom Report Builder launch URL in the Site Settings section of the Report Manager. You can find the report builder application under the Program Files\Microsoft SQL Server\MSRS10_50.MSSQLSERVER\Reporting Services\ReportServer\ReportBuilder folder. To follow along, we are going to set the launch URL in the Report Manager to use Report Builder 1.0. To do this, open up Report Manager by navigating to <http://ServerName/Reports>. Once Report Manager is open, click on the Site Settings link in the top right corner. Set the launch URL to:

<http://localhost/ReportServer/ReportBuilder/ReportBuilder.application>

After you change the default Report Builder URL, now let's use it. Click Apply to save your changes and navigate to the Report Manager home page. Once on the home page, you should see the Report Builder icon on the main toolbar of the Report Manager as shown in Figure 13-20. Click the Report Builder button to initialize the Report Builder 1.0 installation if it is not already installed or to open the installed application.

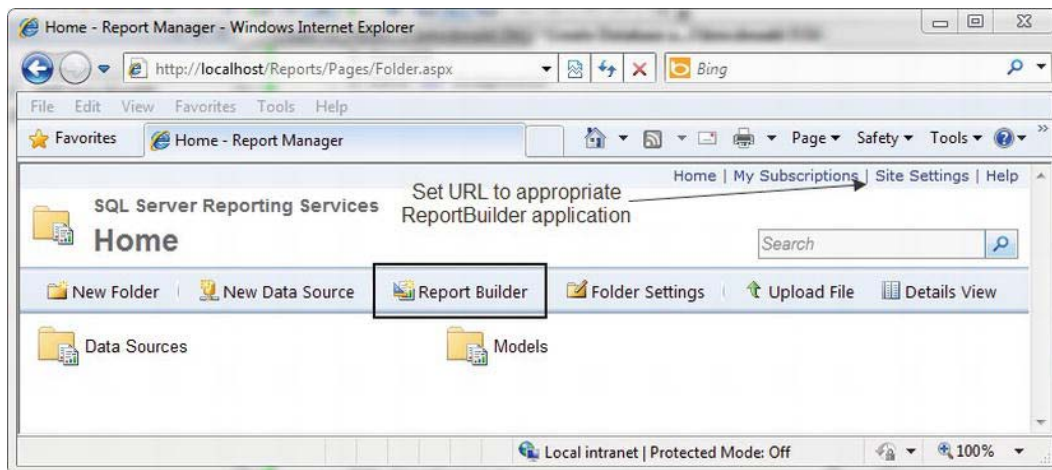


Figure 13-20. Report Manager main toolbar with Report Builder button

■ **Note** As we stated previously, it is also possible to launch Report Builder 1.0 directly from the ReportServer URL such as <https://mySSRSserver/ReportServer/ReportBuilder/ReportBuilder.application>.

Report Builder 1.0 uses ClickOnce technology to install itself from the Web site. The code is downloaded and installed to the local machine from the browser, assuming there are no issues with missing prerequisites, one of which is .NET Framework 2.0. If this is not installed, it will issue a warning message that Report Builder 1.0 requires .NET Framework 2.0 before it can be installed. If all the

requirements are met, Report Builder 1.0 initiates and completes installation. Figure 13-21 shows the ClickOnce application installation screen.



Figure 13-21. Report Builder 1.0 installation progress

After Report Builder 1.0 is installed, it launches automatically. The first step to begin designing a report is to select a source of data, as shown in Figure 13-22. The source will be a report model, in this case the Patient Census report model you have already published. So, select it, and click OK to continue.

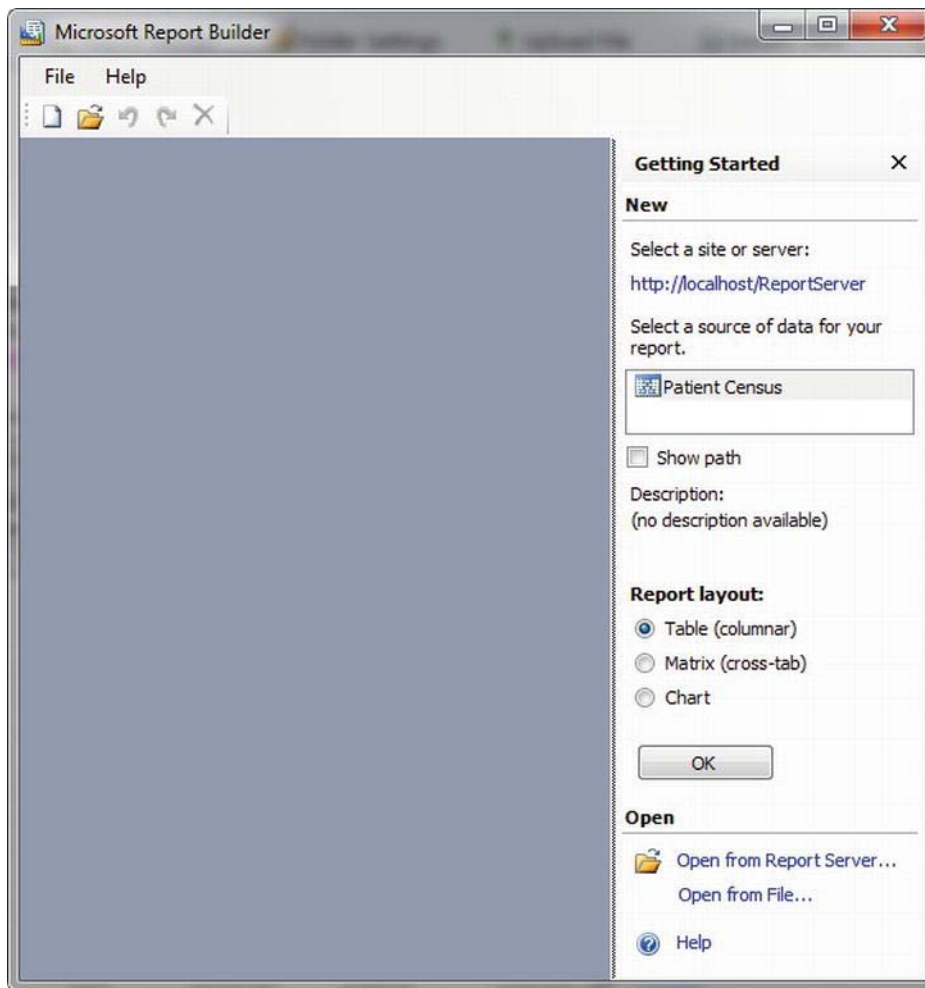


Figure 13-22. *Selecting the report model*

At first glance, it is easy to see that Report Builder 1.0 has the look and feel of other Microsoft design products in the Office 2007 suite, such as FrontPage. You will see a simple design area, a Report Data area, and a Report Layout area that contains templates that can be used for each report. The three available templates are familiar versions of data regions available for SSRS reports: Table, Matrix, and Chart. Each template has predefined areas such as a title, a total column, and a filter description column, as shown in the table report in Figure 13-23. You can also see the fields of the Patient Census report model in the Report Data tab on the left.

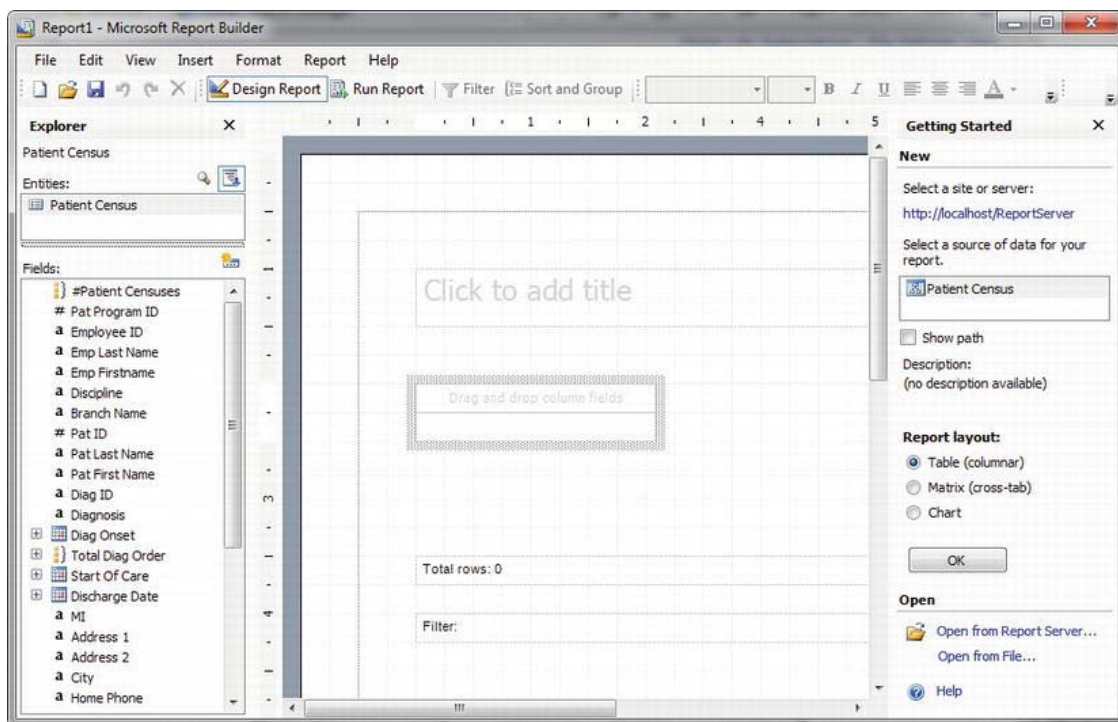


Figure 13-23. Table report template in Report Builder 1.0

Creating a Table Report

To address many of the report requests received from user feedback, you can use the table report. One item to note, which may fall into the category of a limitation of Report Builder 1.0, is that you can use only one data region per report. In other words, unlike the full IDE of BIDS, users who build reports with Report Builder 1.0 are limited to one Table, Matrix, or Chart data region per report. You cannot add a second data region. In fact, reports are not only limited to a single data region, but they are relegated to using the controls that have been defined for each template. It is not possible to add textboxes, for example, or any other type of design element, other than the data from the fields. You will work within this limitation to produce your first report, a *patient face sheet* (as it is commonly called), which displays demographic information about each patient. With the table report still open, you will drag several fields onto the table where it says Drop Column Fields so that the report looks like Figure 13-24. Next, hold the CTRL key down on the keyboard while selecting Pat ID, Pat Last Name, Diagnosis, Start Of Care, Address 1, City, State, and Zip. After you have them all selected, drag them to the right of Pat ID in the table of the report. By holding the CTRL key, as we've done here, we create a group of Patient Census column names. At this point, you can also add a title in the Click to Add Title textbox. Change the title to Patient Face Sheet and make it bold. Notice also that as the fields are added, Pat Last Name, for example, they become bold in the Fields tab on the left indicating they have already been used.

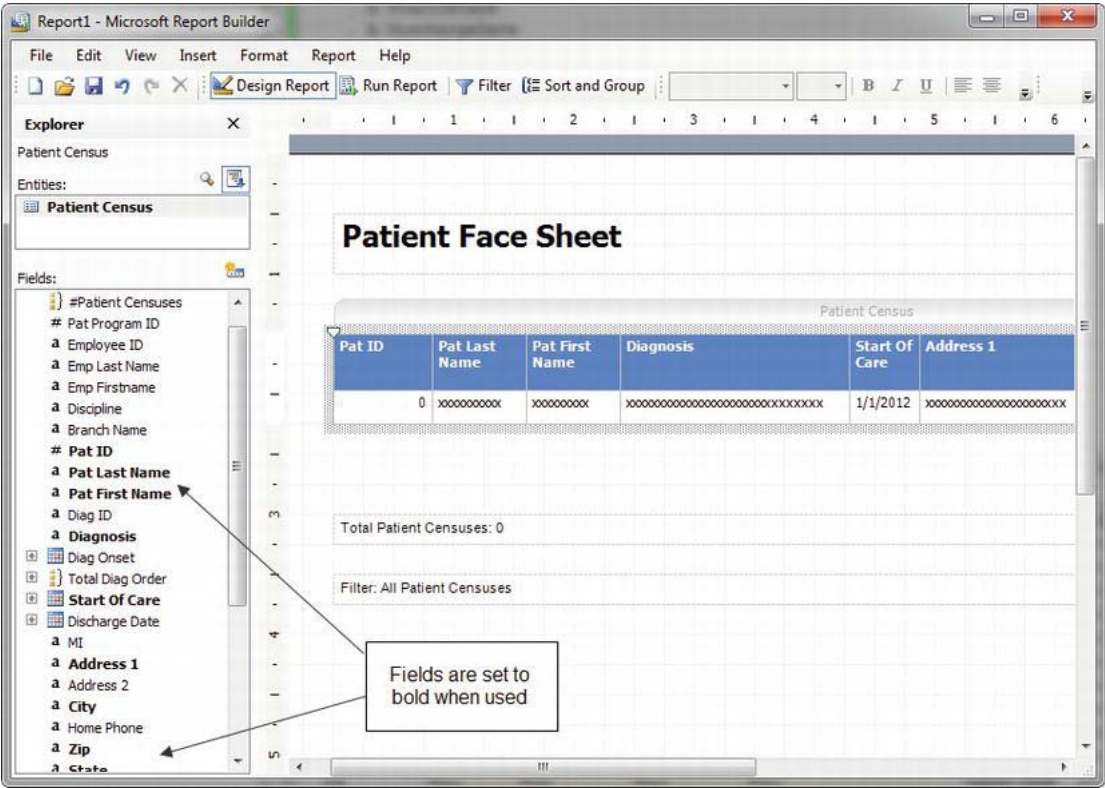


Figure 13-24. Table report with patient information

Once you have added all these fields to the report, you could click Run Report to preview the report; however, at this point, you must perform some tasks to make the report fit on a printed page. In its default page setup, the report is set to Portrait; however, the fields you have added extend beyond 8.5 inches across. To fix this, right-click in the design area, and select Page Setup so that the Page Setup dialog box appears, as shown in Figure 13-25. Set the Orientation property to Landscape, then set all of the Margins to 0 and click OK.

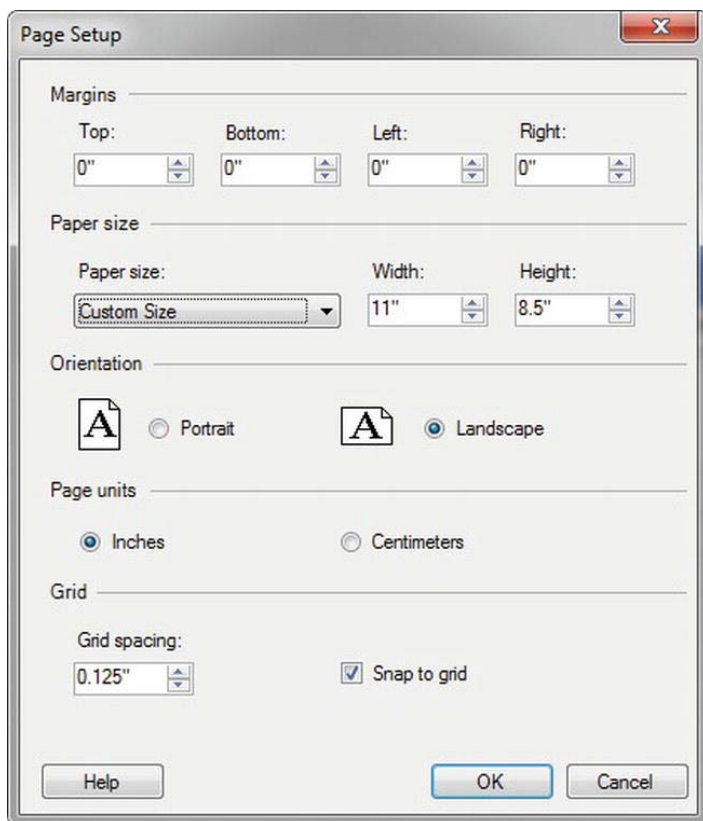


Figure 13-25. Setting the page orientation

To make even more room, you can use a formula to combine the patient's first and last names. To do this, simply right-click the Pat First Name data cell, and select Edit Formula. This will open the Define Formula window. Formulas in Report Builder 1.0 are like expressions in BIDS. Here is where you can combine report fields and built-in functions to produce the desired value. Figure 13-26 shows the Define Formula dialog box with several available functions. Notice the formula used to concatenate the Pat First Name and Pat Last Name fields, which uses report data fields and an ampersand (&) to add a literal comma character to separate the names. You can also use the RTRIM function to remove trailing spaces. You could have done much of this work in the report model, where it most likely should have been done. However, we had you leave the names this way so we could now demonstrate the use of a simple formula. Once the formula is in place, click the Save this formula as a new Patient Census field, and click OK. You are then prompted to enter the New Field Name. Call this field Patient Name and click OK to save your new field.

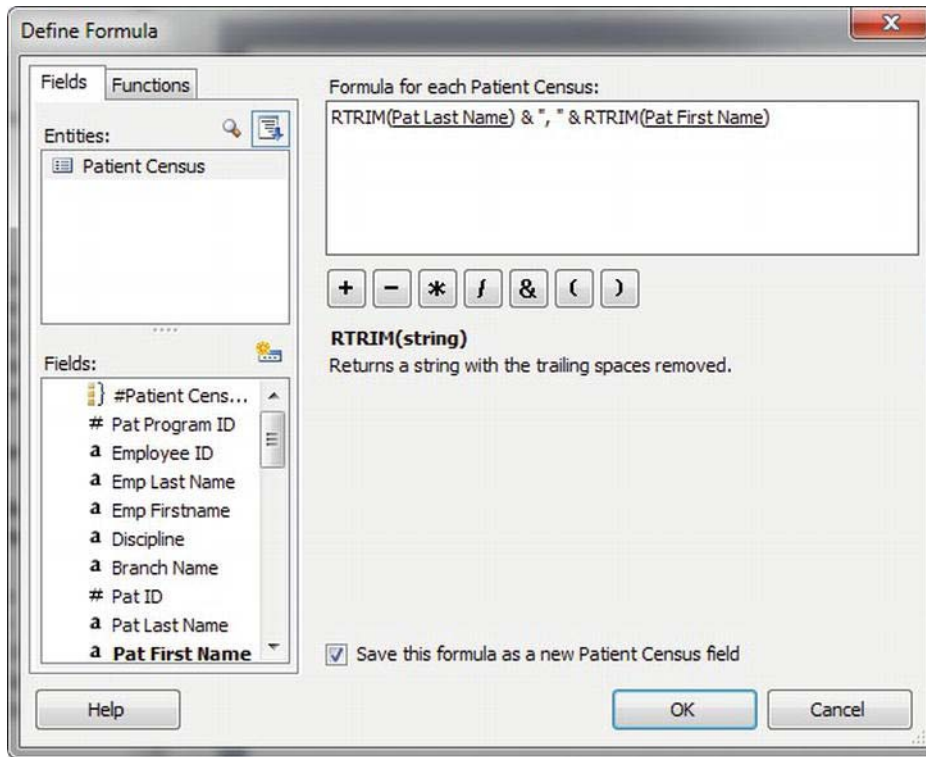


Figure 13-26. Define Formula window

Now that you have the patient's full name combined, you can delete the Pat Last Name field by clicking the data cell and pressing the Delete key. A couple of other tasks remain before previewing the report. The first one is to move the Zip column to the right of the State column. To do this, just click on the cell containing Zip and drag it to the right of the State column. Next, let's resize the fields so that each record will fit on a single line. Of the fields you have used for the report, the Diagnosis and Address fields have the potential to be the largest, as they are variable-length fields that usually contain between 20 and 50 characters. However, since we want the headers for each column to be on one row, we will also resize Start of Care. For this example, size the Pat ID field to about .5 inches, expand the Diagnosis, Start of Care, and Address 1 fields to approximately 1.5 inches, and preview the report, as shown in Figure 13-27. There will be as many *x* characters representing the data as there are characters found in the largest value in the database field, which makes it easy to determine how much to resize each column. The reason that the Address fields display NA is because this information was either scrambled, in the case of the patient names, or modified to be NA to remove identifiable information. Notice also, in Figure 13-27, that each field automatically contains an interactive sorting icon at the column heading level. Though clicking the icon will re-render the report, no actual sorting will be performed until you configure it to do so. You will do that next.

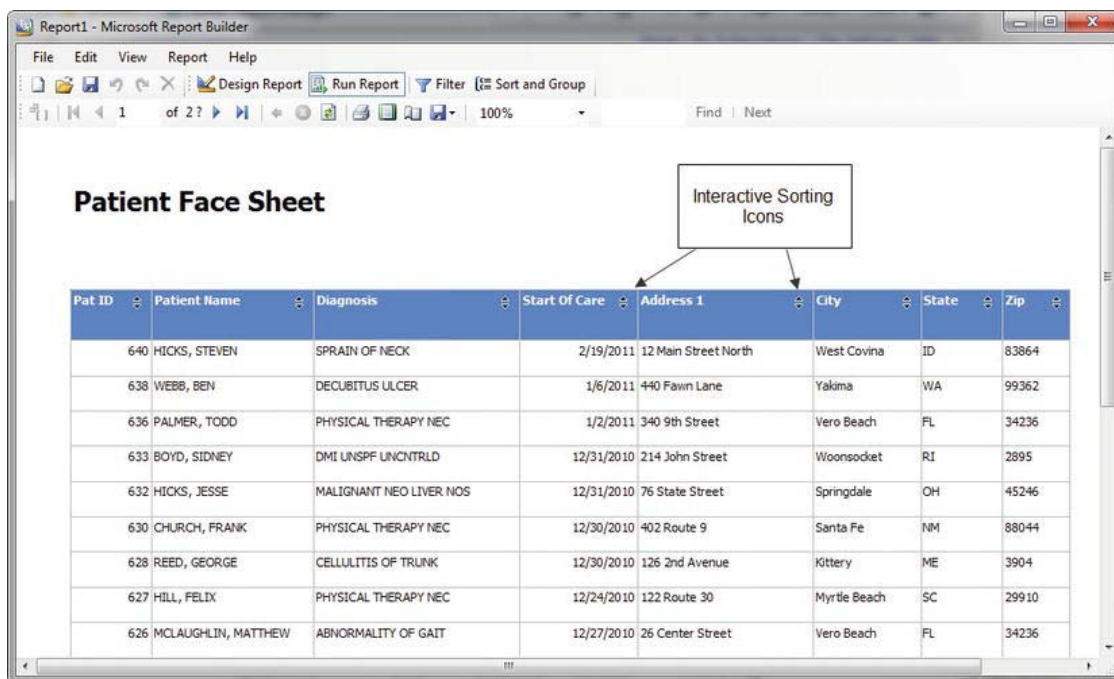


Figure 13-27. Previewing the Patient Face Sheet report

Return to the design area of the report by clicking the Design Report button on the toolbar. Next, right-click the Pat ID data cell, and then select Sort and Group on the toolbar. This will open the Sort dialog box. With the Patient Census group selected, choose to first sort by Pat ID, then by Patient Name, and finally by Diagnosis, as shown in Figure 13-28. You also have the ability to put a page break after each group as a group option. Do not add a page break at this time. Once you click OK, you can preview the report and see that the sorting for each of the defined columns is working correctly.

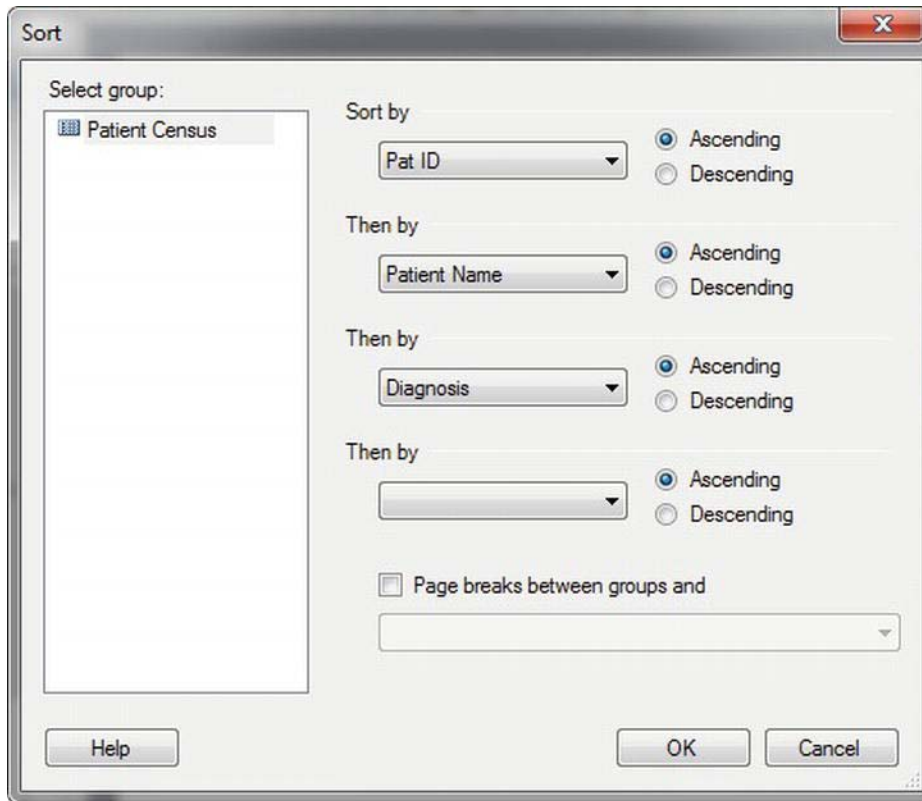


Figure 13-28. Sorting and grouping options

We should mention one more thing about dragging fields to the table area of the report: groups are automatically defined from left to right. In other words, dragging and dropping fields to the right of another field keeps the field in the same group, in this case the Patient Census group; however, if you have already dragged a field to the table and then drag another one to the left of it, another group is created based on the name of the field that is dropped. You can see this in Figure 13-29, where if you were to drag the Branch Name field to the left of the Pat ID field. If you are going through these steps, go ahead and drag the Branch Name field to the report to see the automatic grouping. Also, go ahead and resize the table header row to have a height of one line. Then drag the entire table to be a little closer to the title. Performing these steps should result in a report similar to that shown in Figure 13-29.

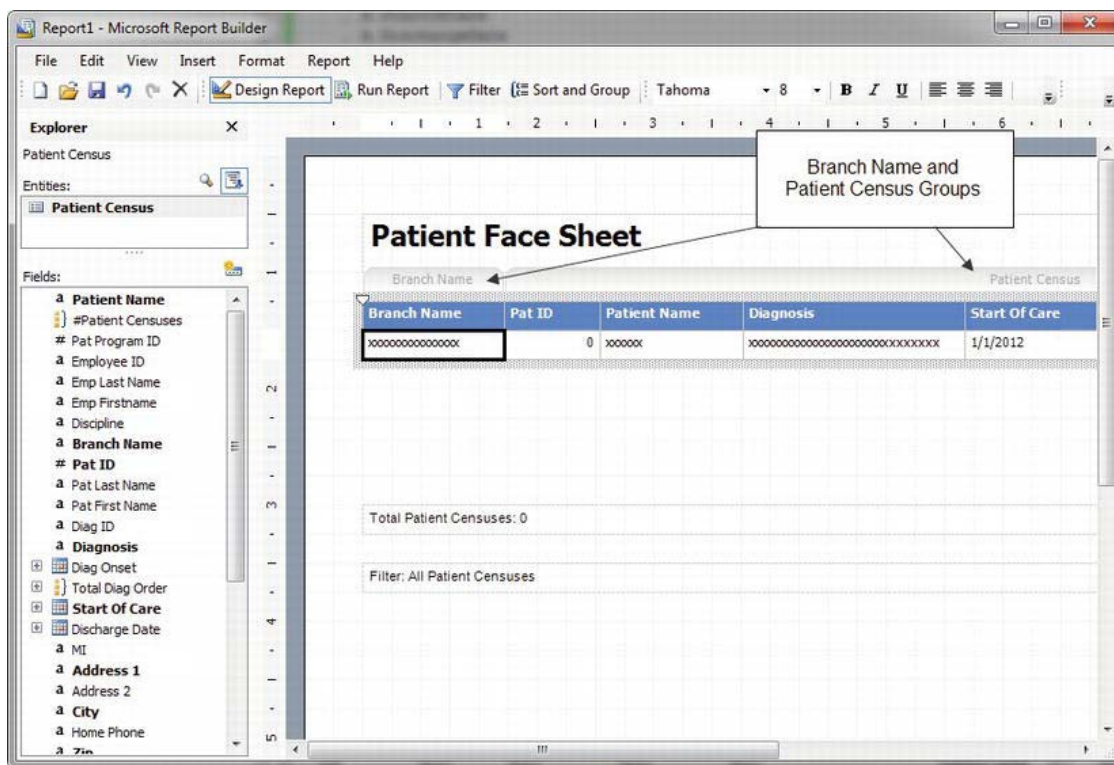


Figure 13-29. Automatically created groups

You can preview the report one last time before saving it to the report server. Notice in Figure 13-30 that the newly added branch group hides duplicate values automatically and groups each record from the Patient Census group under its assigned branch name. You could, at this point, use standard formatting features such as adding a border or changing the background color of cells or even the font size, color, and justification if you so desired, but the base report meets your needs, so go ahead and deploy it.

Branch Name	Pat ID	Patient Name	Diagnosis	Start Of Care	Address 1	City
Fountain Square	501	LIVINGSTON, JAVIER	COR ATH UNSP VSL NTV/GFT	10/22/2010	192 Vine Street	Mill Valley
	502	CARPENTER, RODNEY	CHR AIRWAY OBSTRUCT NEC	11/25/2010	102 Route 1	Sunset
	502	CARPENTER, RODNEY	PHYSICAL THERAPY NEC	10/23/2010	102 Route 1	Sunset
	505	MORRISON, JONATHAN	DECUBITUS ULCER	10/26/2010	414 Ashley Court	Sunset
	505	MORRISON, JONATHAN	DECUBITUS ULCER	9/27/2010	414 Ashley Court	Sunset
	506	VANCE, LEON	CVA	11/30/2010	496 Westminster Drive	Sunset
	506	VANCE, LEON	LATE EFFECT CV DIS NOS	8/1/2010	496 Westminster Drive	Sunset
	507	PATTERSON, RONALD	DECUBITUS ULCER	9/26/2010	242 Devon Road	Missoula
	508	SPENCE, RONALD	DMI UNSPF UNCNRD	10/25/2010	252 Laurel Lane	Sunset
	509	FOSTER, TRACY	ATRIAL FIBRILLATION	9/28/2010	360 Elm Avenue	Santa Fe
Fountain Square	510	PUGH, MICHAEL	PHYSICAL THERAPY NEC	12/16/2010	350 Buttonwood Drive	Sunset
	510	PUGH, MICHAEL	PHYSICAL THERAPY NEC	10/27/2010	350 Buttonwood Drive	Sunset

Figure 13-30. Previewing a new Branch Name group

Deploying the report to the report server is as simple as clicking the Save button on the toolbar. After clicking Save, a Save Report dialog box appears that allows you to navigate to an accessible folder on the report server; in this case, open Pro_SSRS folder by double clicking it. If you do not have a Pro_SSRS folder on your SSRS 2008 R2 server, simply navigate to the Report Manager (<http://localhost/Reports/>) and click the New Folder button on the toolbar of the home screen. Name the folder Pro_SSRS and click OK to create the definition of the new folder in the ReportServer database. Name this report Patient Face Sheet, as shown in Figure 13-31, and click the Save button.

You can navigate to the Report Manager later to make sure the report was saved successfully, but for now, you will move on to the next type of report you will learn how to create, a matrix report.

■ **Note** Though you will save the report to the report server and can load it back into Report Builder 1.0 for modification, it is possible to open a Report Builder 1.0 RDL file from a file location as well. Opening a report from a file is a choice in the task pane, accessible by selecting the Task Pane under the View menu or by pressing Ctrl+F1.

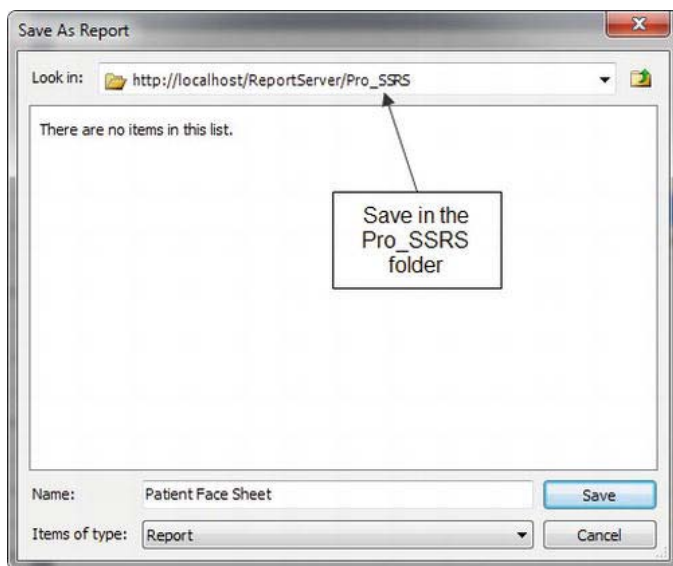


Figure 13-31. Saving a report to the server

Creating a Matrix Report

Unlike a table report, a matrix report displays aggregated values two-dimensionally with column groups as well as row groups. The totals, whether they are sums, averages, or counts, intersect at the grouping levels for columns and rows. This creates a cross-tab report similar to a pivot table in Microsoft Excel. One of the requests for the report was to show the length of stay for patients, from the time they were admitted to the time they were discharged. It is also a requirement that the discharge reason show on the report. You can combine these two requests perfectly into a single matrix report.

So, return to the Report Builder 1.0 design area, and click the Matrix (cross-tab) report in the Report layout section of the Task Pane. This opens a new blank matrix template where you can drag and drop fields onto the Matrix data region, just as you did with the table report. The template is nothing more than a predefined Matrix data region to which you will add several fields, as shown in Figure 13-32.

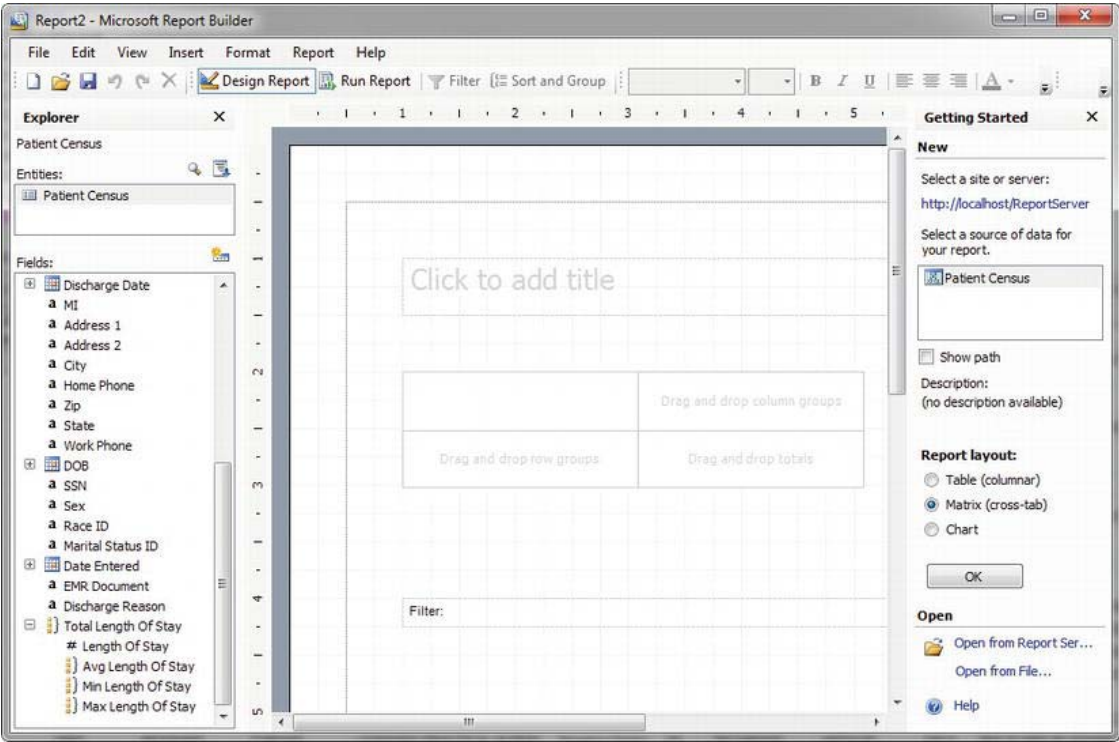


Figure 13-32. Blank matrix report

To start this report, expand the Total Length of Stay attribute in our Fields list. Drag the Avg Length of Stay attribute that was calculated when the model was generated over to the totals area of the Matrix. For the row grouping, add the Discharge Reason field. Finally, drag the Marital Status ID field to the column grouping area so that the matrix report resembles Figure 13-33. It will be necessary to resize the Discharge Reason field and center-justify the Marital Status ID field. You will also at this time add a title to the report: Average Length of Stay. Make the title bold and set the font type to Arial Rounded MT Bold.

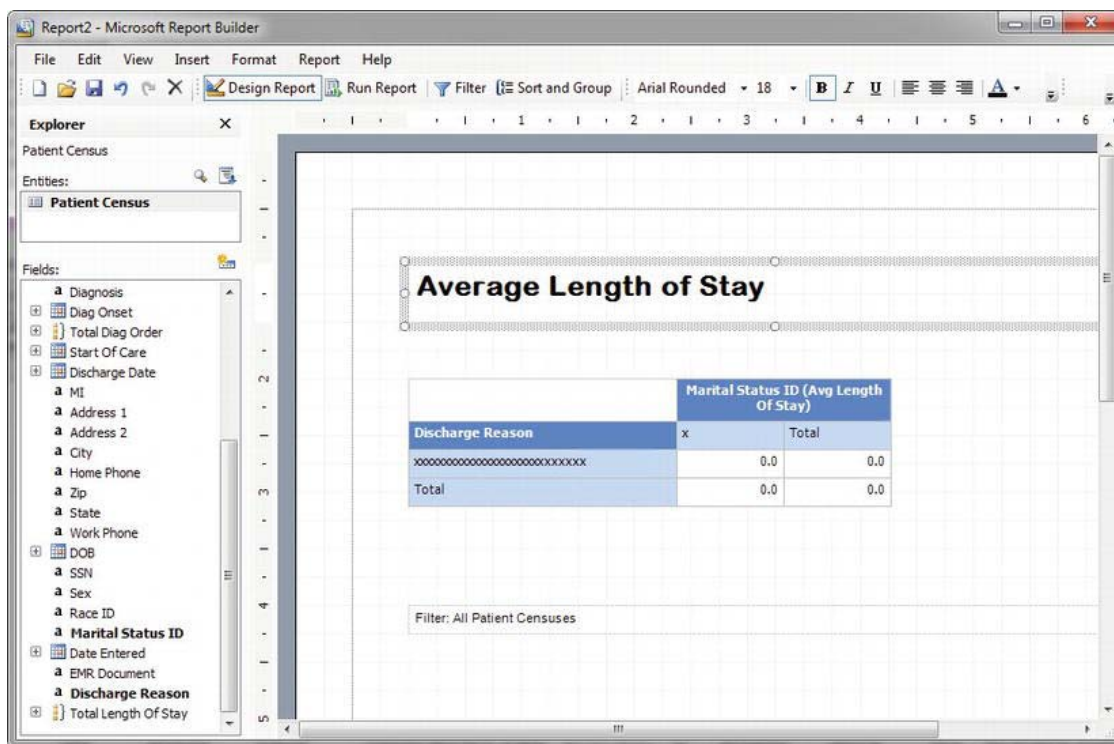


Figure 13-33. Matrix report with data

When previewed, the report is already taking shape, as shown in Figure 13-34; however, you need to address some anomalies. First, the number format is not what you expected or wanted. A general number, which by default is the data type assigned to the Avg Length of Stay field, is not desired. You will need to add formatting to the number. Second, you will see rows and columns with empty values. This is happening for two likely reasons. First, you are including patients who have not been discharged yet, in other words, active patients. Second, marital status is not a required field, so some records will not have values.

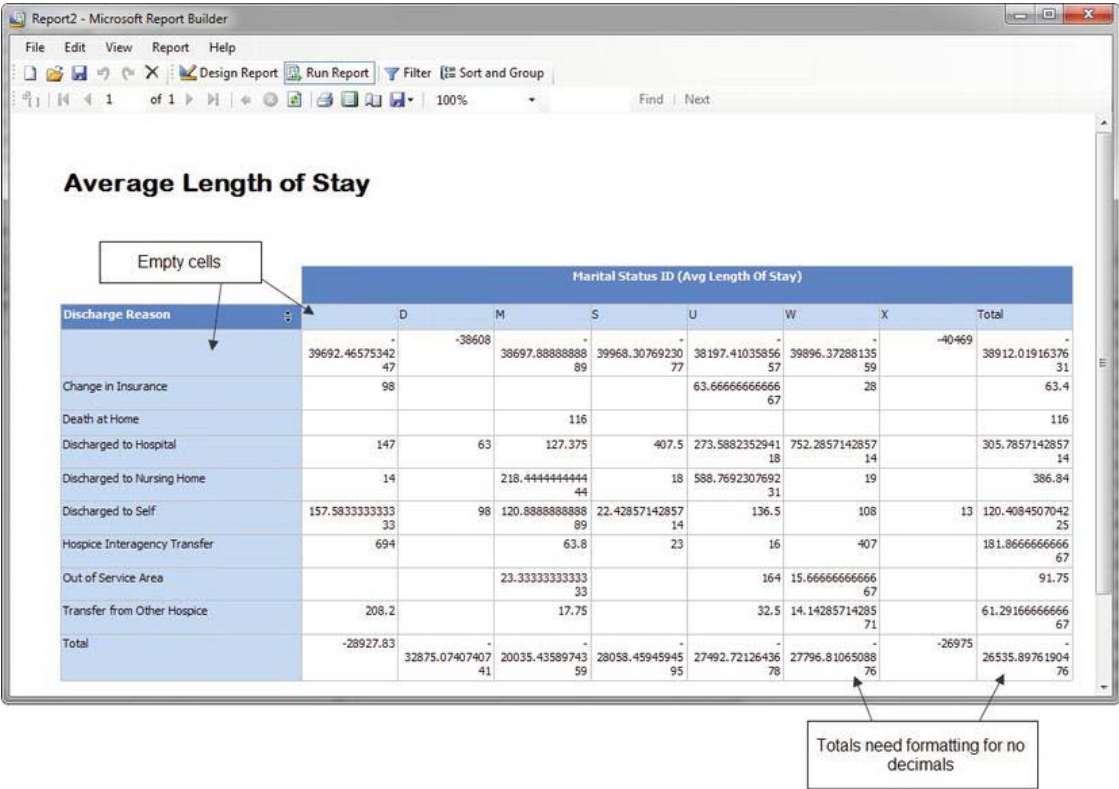


Figure 13-34. Initial preview of a matrix report

You can take care of the number formatting by returning to the design area, right-clicking the last column (which is the total), and selecting Format. In the Format dialog box, change the defined format from General to 1234.56, as shown in Figure 13-35, and change the decimal places from 2 to 0. Then click OK. Perform the same formatting on all of the other value fields.

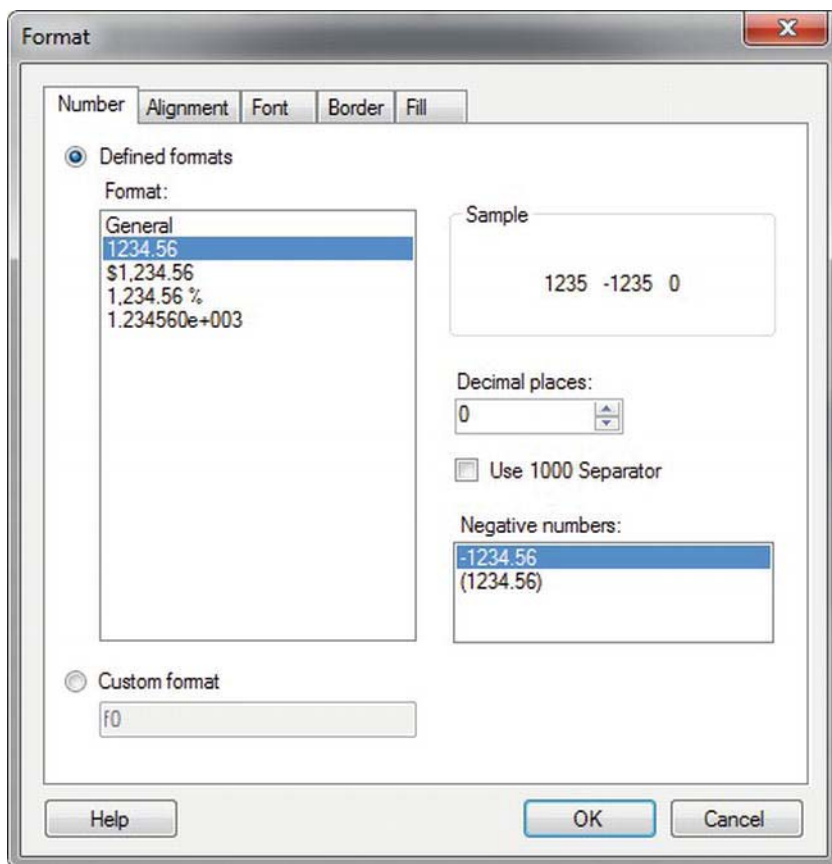


Figure 13-35. Changing the number formatting

To address the empty columns, add a filter to the report to limit the data to only discharged patients with discharge reasons and to patients who have a marital status defined. The Filter button on the toolbar will launch the Filter Data dialog box. In Figure 13-36, you can see that you can add four filter conditions: where MaritalStatusID, Discharge Date, and Discharge Reason are not empty and where the Discharge Year is greater than or equal to 2010. (We added the last filter to demonstrate that it is possible to have the report prompt the user for a value as a parameter that is tied to a filter.) To change the comparison operator from the default of equal to not empty, click on empty and select both Not and Is Empty from the available operators. For the Discharge year, select the Greater Than or Equal To operator. After you set the year to be 2010, click on the Discharge Year label and select Prompt to set it as a report parameter as depicted by the green question mark icon shown next to the Discharge Year filter label.

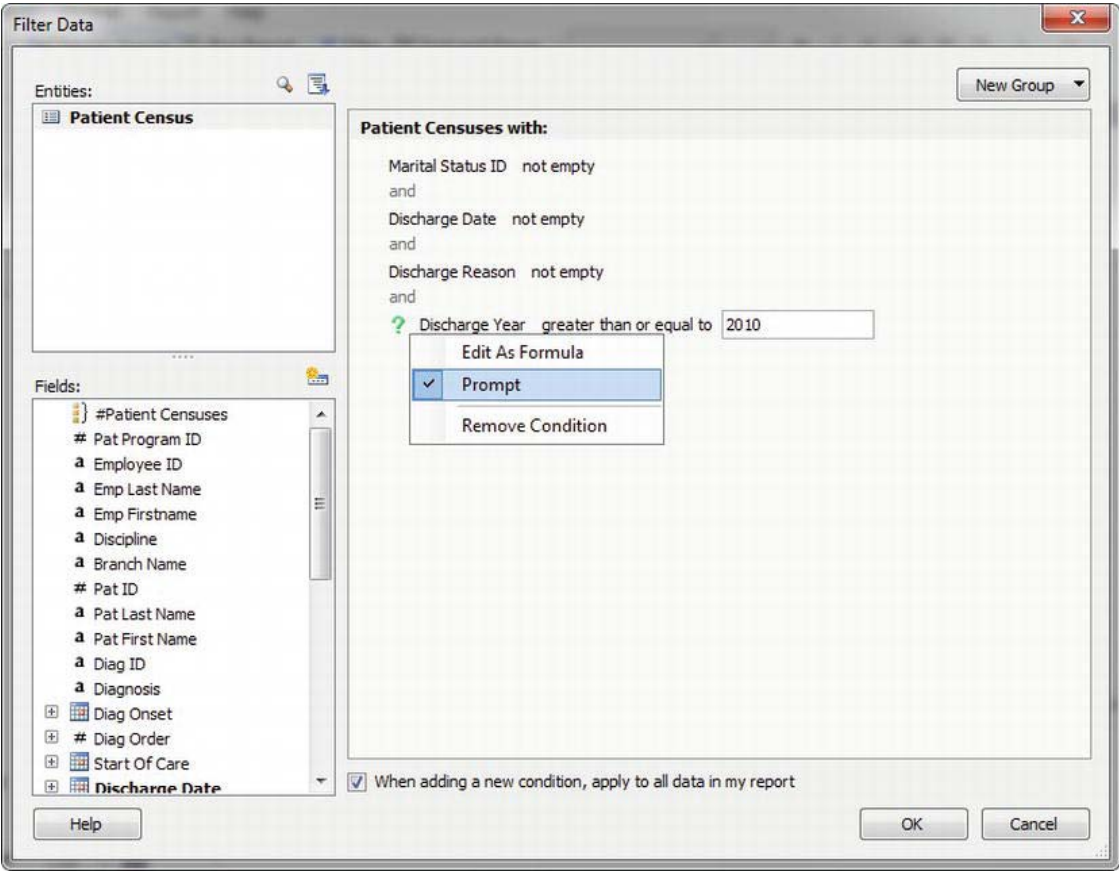


Figure 13-36. Setting up report filters

Now, when you preview the matrix report in Figure 13-37, you can see that the numbers have been cleaned up and all empty fields are gone; also, you are prompted for the discharge year. Change the Discharge Year parameter to 2011 to see those results. Save the report as Average Length of Stay to the Pro_SSRS folder on the report server, just as you did the table report. We will now move on to create the final report using Report Builder 1.0, the chart.

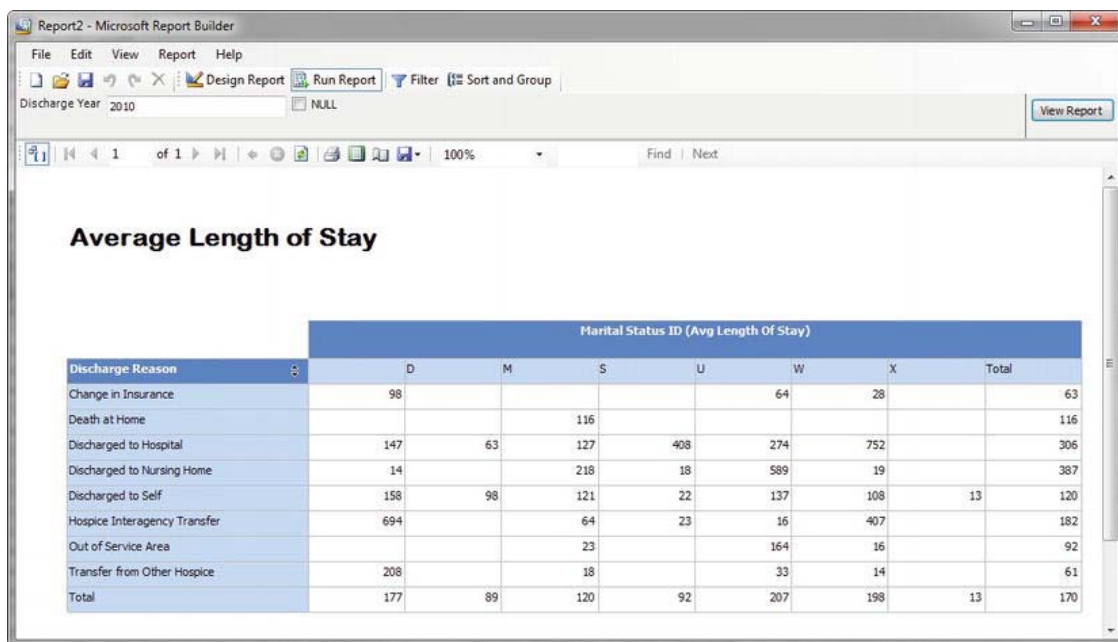


Figure 13-37. Setting up report filters previewing the report

Creating a Chart Report

The last report type available in Report Builder 1.0 is the chart report. Thus far, you have created and saved table and matrix reports to address many of the report requests for the Patient Census report. Now, we will show one of the most important reports to answer the question of the number of patients on the census. Census means that the patients are currently receiving services and have not been discharged. In the database you are using, a patient is considered active if his or her discharge date is Null, or, in Report Builder 1.0 vernacular, *empty*. The chart you create will be a simple bar chart showing a count of active patients by their branch association.

Return to the design area of Report Builder 1.0, and click the chart report in the Report Layout tab. The blank report template has a single Chart data region, as shown in Figure 13-38. As you did with the two previous reports, table and matrix, you will add a title: Active Patient Census.

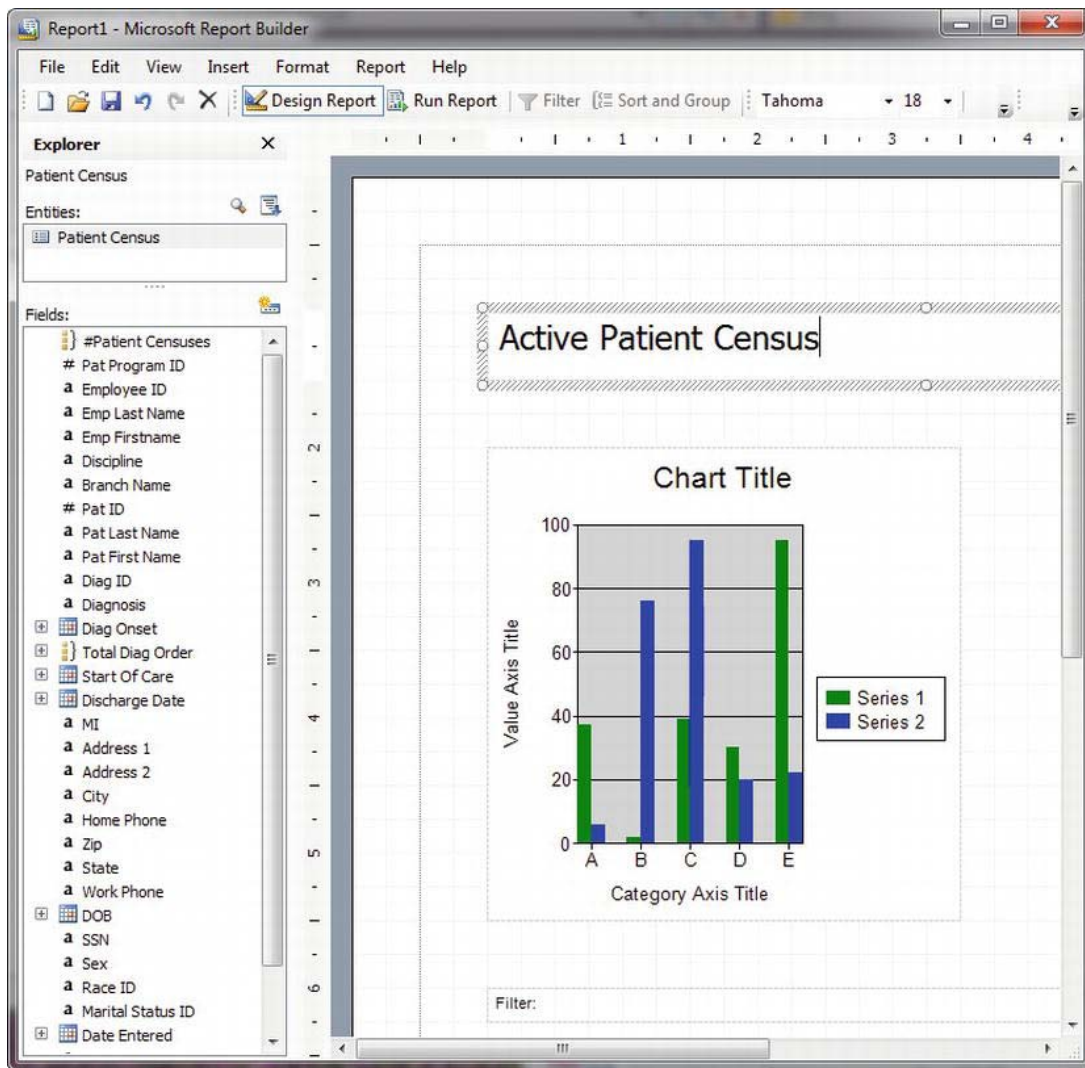


Figure 13-38. Chart report template

The chart has three areas to which you can drag and drop data: Series, Category, and Data. You want to be able to show an active patient count, so you could drag the Pat ID field to the Data area; however, because the Pat ID field is not seen as a field that can have an aggregated function applied to it, unlike Avg Length of Stay, you cannot drag it directly to the Data area. The chart simply won't accept it to be dropped there. Since you know you will need both a count value and a filter to get only active patients, you can take care of both at the same time by adding a new field to the field list. Click the New Field button in the Explorer tab, and name the new field Patient Count, as shown in Figure 13-39. The formula you can use for the Patient Count field is `COUNTDISTINCT(Pat ID)`. Notice that in the Define Formula

dialog box that the Pat ID field, when added to the Formula textbox, has a dotted line beneath it indicating a link. You can click the field and add a filter while defining the formula for the new field, which you will do next.

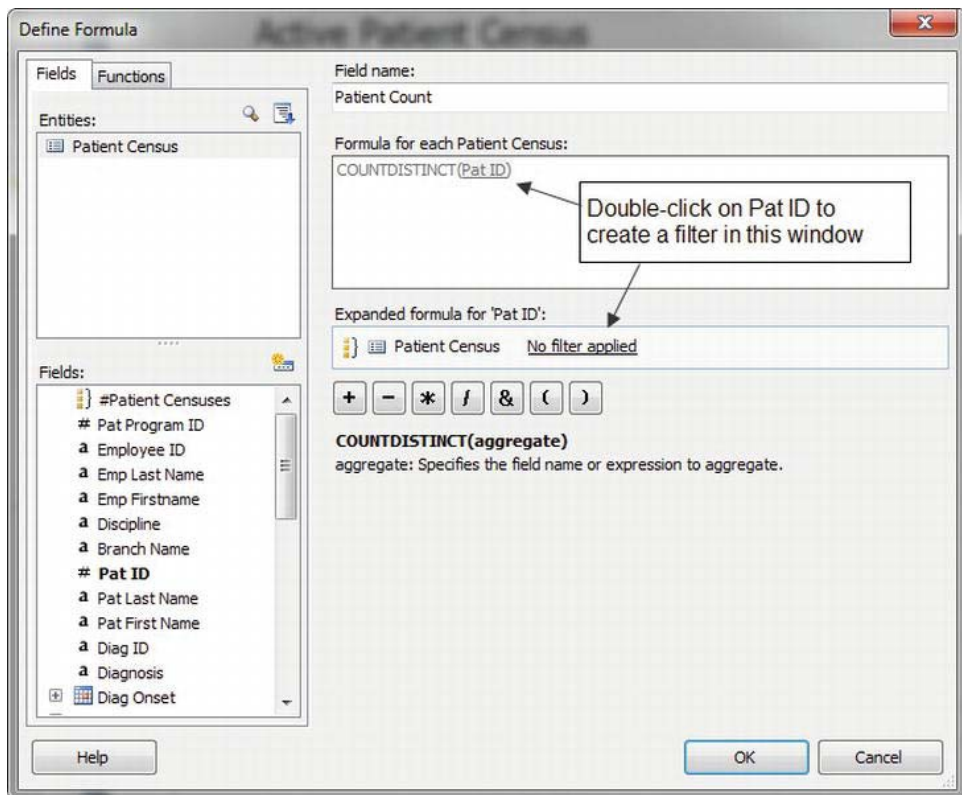


Figure 13-39. Defining a formula for a new field, Patient Count

Click the No Filter Applied link, and select Create a New Filter to open the Filter Data dialog box. You can see the filter you need to apply to show only active patients in Figure 13-40, where you have set Discharge Date to “empty,” which will force only patients who do not have discharge dates to be used in the formula. To implement the filter, drag Discharge Date over into the filter pane. Then select the Is Empty value by clicking on the equals link label. Call the filter Active Patients, and click OK. Once the Patient Count field is created and shows up on the Fields tab, you can drag and drop it in the Data value fields area of the chart. In this case, since we are adding it to the Data value fields area, we could simply double-click on the Patient Count field and Report Builder will place it in the correct section.

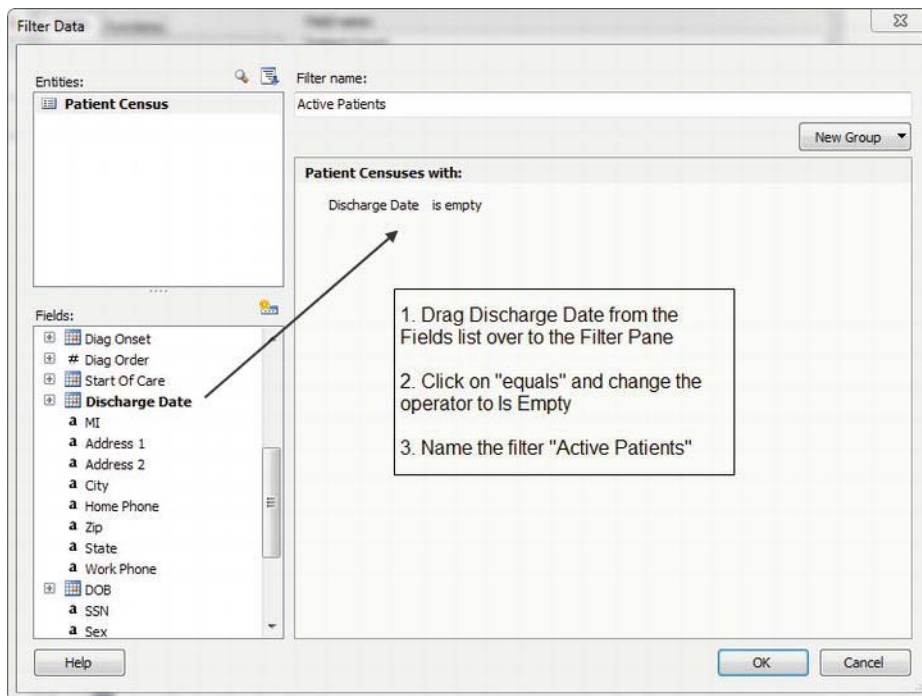


Figure 13-40. Adding the Active Patients filter

Next, drag the Branch Name field to the Categories area of the chart. You could add further series items or categories to the chart, but to keep the chart simple and to address the request, you will just apply some formatting to the chart by way of removing the chart title so that it does not say “Chart Title” on the report. To modify the chart properties, follow these four steps:

1. Right-click the chart, and select Chart Options.
2. On the Titles tab, delete the text for the chart, Category X and ValueY titles.
3. On the Legend tab, remove the check from the Show legend checkbox.
4. Finally, on the 3D tab, check the Display Chart with 3-D Visual Effect box, set Horizontal Rotation to 25, set Vertical Rotation to 10, Realistic as the Shading, and select Cylinder as the shape, as shown in Figure 13-41.

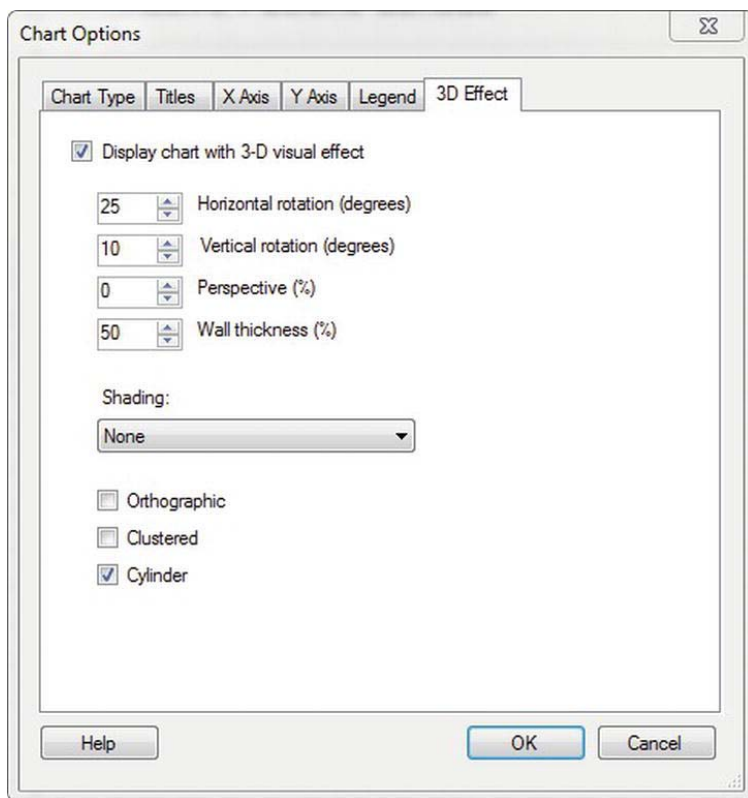


Figure 13-41. Setting 3D visual effects

The last task before previewing the report is to set the data labels on the chart. To do this:

1. Right-click the chart, select Data Series, and then select Format Data Series—Patient Count. If you would like to have the patient count value display on the bars of the chart, select Show Point Labels. For this example, select the option to Show point labels.
2. Set the angle to –10 degrees to make it stand out a little.
3. Click the Format button to set the font size to be 16.

When you click OK to set the point labels and then preview the report, you can see the chart with the active patient counts per branch. As patients are discharged, they will drop from the active patient list because of the filter, and the chart will automatically reflect the current census count. Figure 13-42 shows the completed chart. Now you can save the chart report as Active Patient Census to the Pro_SSRS folder, as you did the other two reports.

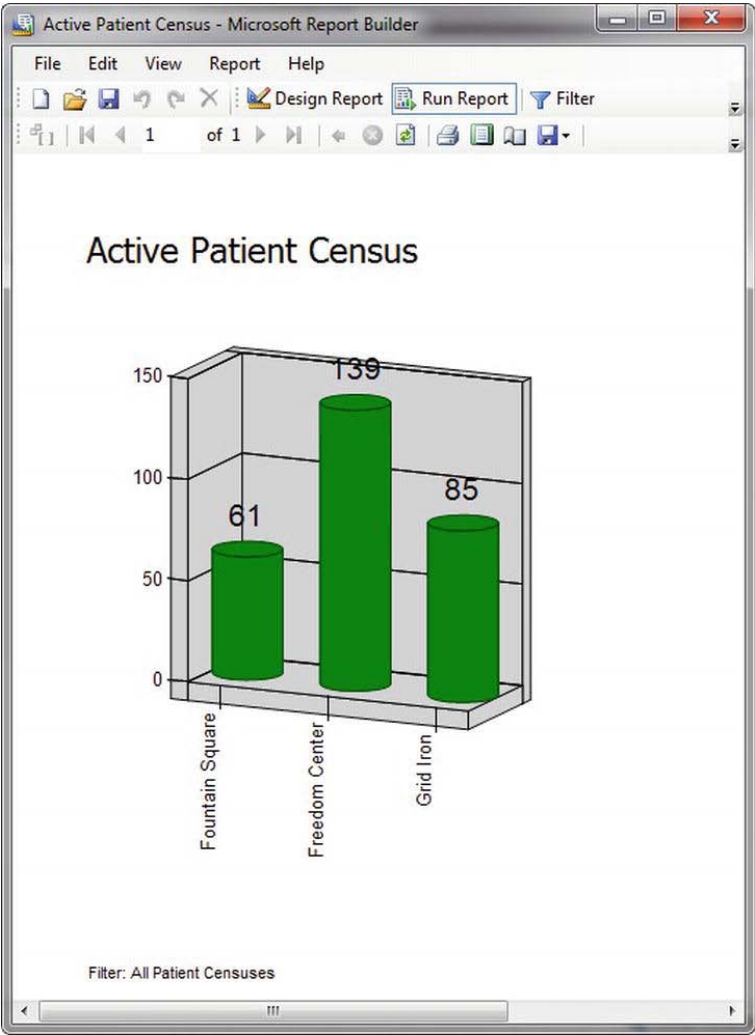


Figure 13-42. Preview of the Active Patient Census chart report

The reports are deployed as standard RDL report files with the exception that their data source is a report model and not a standard RDS data source object. You can see the three newly published reports in Report Manager, as shown in Figure 13-43.

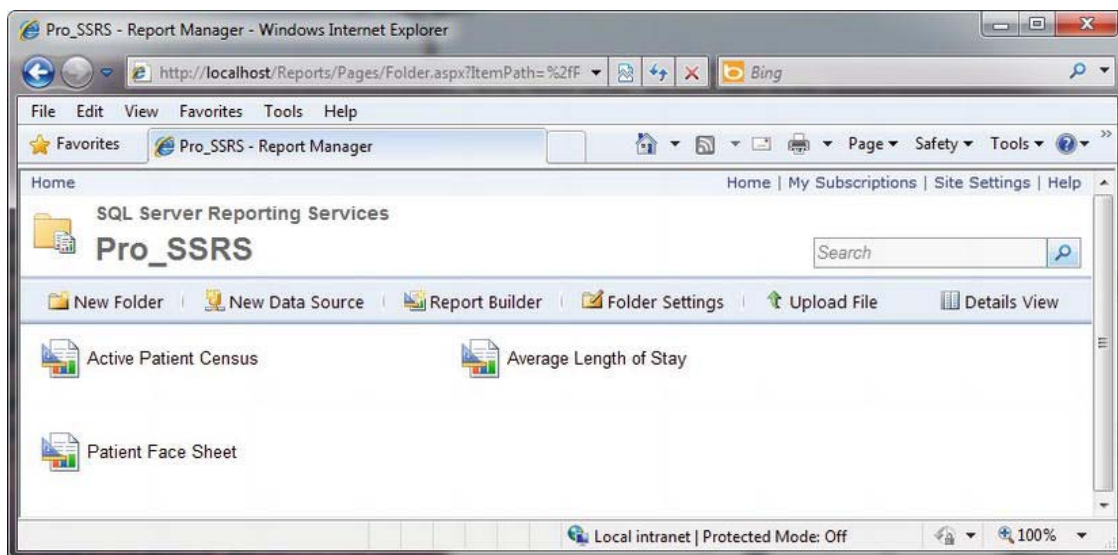


Figure 13-43. Report Builder 1.0 reports published in Report Manager

Creating Reports with the Report Builder 2.0 Wizard

At this point, we created SSRS reports with BIDS and Report Builder 1.0 using ClickOnce. Both design environments provide different levels of control over report design. While BIDS offers report creators much more flexibility than Report Builder 1.0, it might not be the best choice for people who are not also application programmers. Report Builder 1.0, by contrast, is easy to learn and to use to publish reports; however, it comes with limitations in that reports can only be created using prebuilt models, which is not enough to meet everyone's needs. In SSRS 2008, Microsoft provided Report Builder 2.0, which supported true ad hoc query and report design not solely based on report models. Report Builder 2.0's modern look and feel, with Ribbon technology like that found in Office 2007 applications, gave report creators a familiar environment to work in.

Figure 13-44 shows the Report Builder 2.0 interface. Microsoft's use of the new Ribbon technology is apparent in the figure.

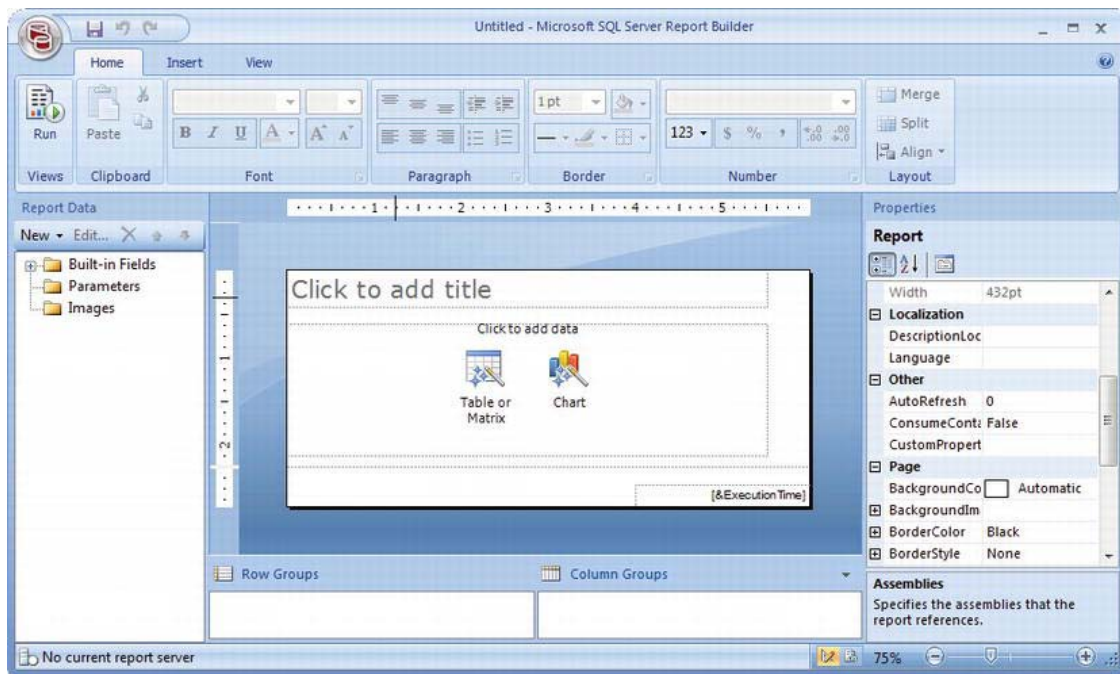


Figure 13-44. Report Builder 2.0's new interface

■ **Note** In order to use Report Builder 2.0 on a SQL Server 2008 R2 or SQL Server 2012 box, you will need to install the Report Builder 2.0 thick client. This client can be downloaded from www.microsoft.com/en-us/download/details.aspx?id=24085. If you are using a SQL Server 2008 SP1 box, you can use the ClickOnce application from the Report Builder button on the Report Manager or from within SharePoint. By default, Report Builder 1.0 is configured within the Report Manager, so you will need to change the Report Builder URL on the Site Settings page to reflect your installation folder. For example, if I had SQL Server 2008 SP1 installed, I would change mine to be `/ReportBuilder/reportbuilder_2_0_0_0.application` or the full URL of `http://ServerName:PortNumber/ReportServer/ReportBuilder/reportbuilder_2_0_0_0.application`. When in SharePointIntegrated mode, you change the Custom Report Builder URL to be a slightly different URL of `/_vti_bin/ReportBuilder/ReportBuilder_2_0_0_0.application`.

As you might imagine, developing a report in Report Builder 2.0 is simplified, with all design choices available in one of three tabs, Home, Insert, and View. Built-in fields, parameters, images, and data sources are all contained in folders within the Data window. Like in SQL Server Data Tools (SSDT), Row Groups and Column Groups areas located are on the bottom. Instructional text, as well as link buttons

that fire up the Table, Matrix and Chart wizards, have been added to the body of the report to guide the designer through the required steps to producing a finished product.

■ **Note** Since the Report Builder 2.0 and Report Builder 3.0 reports are not using a Report Model as a data source, I have installed Report Builder 2.0 on the server containing SQL Server 2012. Being that we have used this server throughout the text in developing our reports, it contains the full Pro_SSRS database.

By way of example, let's build a new version of the Employee Service Cost report that you've been seeing throughout this book. If you are using SQL Server 2008 R2 or SQL Server 2012 and using the Report Builder 2.0 from installing the thick client as I am, you will start Report Builder 2.0 by navigating to the Microsoft SQL Server 2008 Report Builder 2.0 folder under All Programs of Windows start menu. Once you have Report Builder 2.0 fired up, you will want to click on the Table or Matrix icon on the report design surface. As mentioned, this will bring up the New Table or Matrix wizard and begins by prompting you to select an existing data source or create a new one as shown in Figure 13-45.

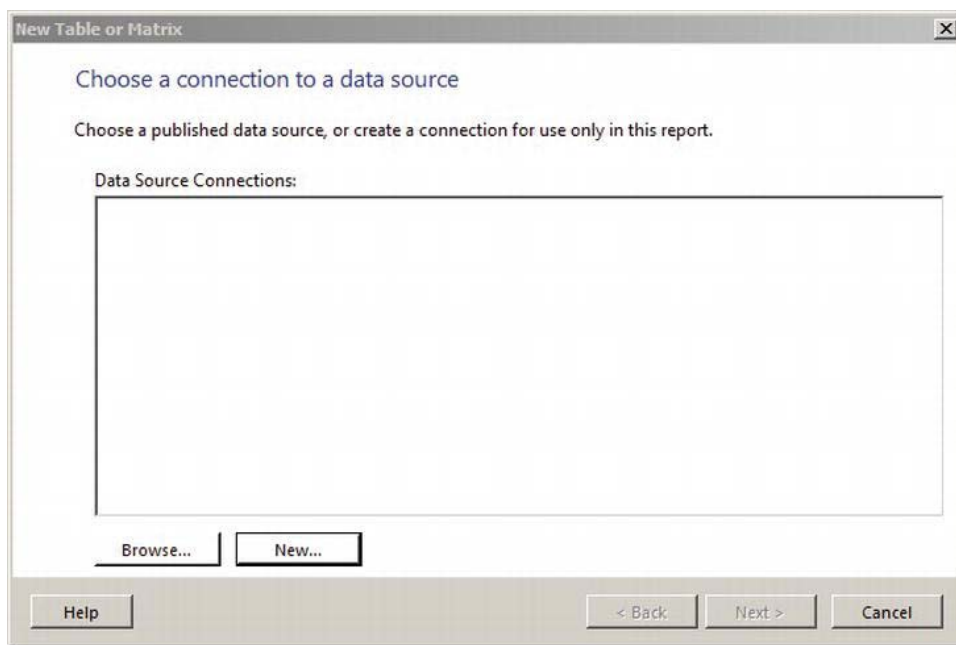


Figure 13-45. *New Table or Matrix wizard in Report Builder 2.0*

Select New... to create a new data source for this report. Name the data source Pro_SSRS and with the connection type set to Microsoft SQL Server, click on the Build... button to select the server. In this case, set the server name to localhost, Pro_SSRS as the database name, as in Figure 13-46. Click OK to save the Connection Properties and then click OK again to save the Data Source Properties. Now that our data source has been created, click Next to build the query in the query designer window.

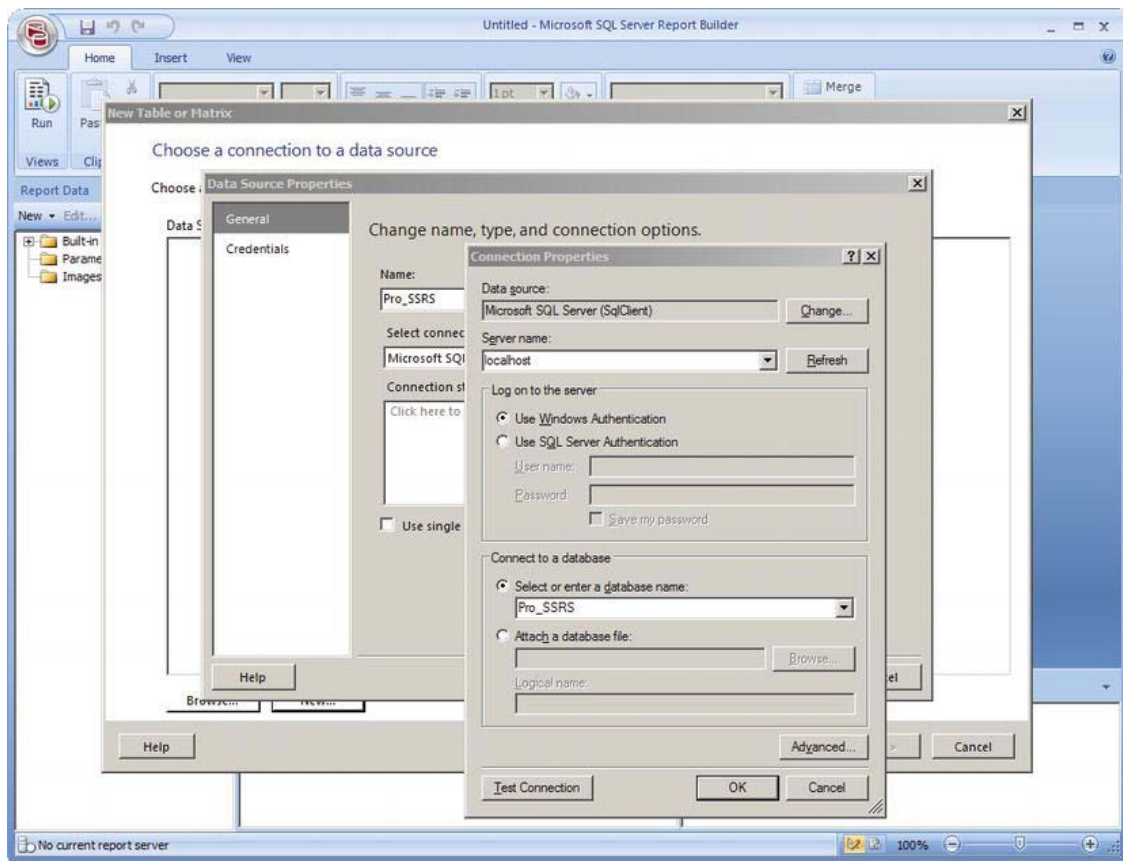


Figure 13-46. Setting data source and connection string properties

The query designer contains a minimal set of tools for query design and is similar in fashion to query builder applications in SSMS. You can graphically add tables to a query, use stored procedures, table valued functions, create joins, and filter data with criteria. In our case, we are going to use the `Emp_Svc_Cost` stored procedure. In the query designer, we can expand Stored Procedures in the Database view pane and then select the `Emp_Svc_Cost` stored procedure. As you can see after selecting the stored procedure, you are shown the fields that the stored procedure returns as well as the parameters that the stored procedure expects. If you want to see the Query results, set the parameter values that you want to set and then run the query. For this example, leave the default values set to (null) and execute the stored procedure by clicking the Run Query button. Figure 13-47 shows the stored procedure `Emp_Svc_Cost` being executed with resultant data. Alternatively, we could have clicked on the Edit As Text button, selected Stored Procedure as the type and then entered `Emp_Svc_Cost` in the query text box.

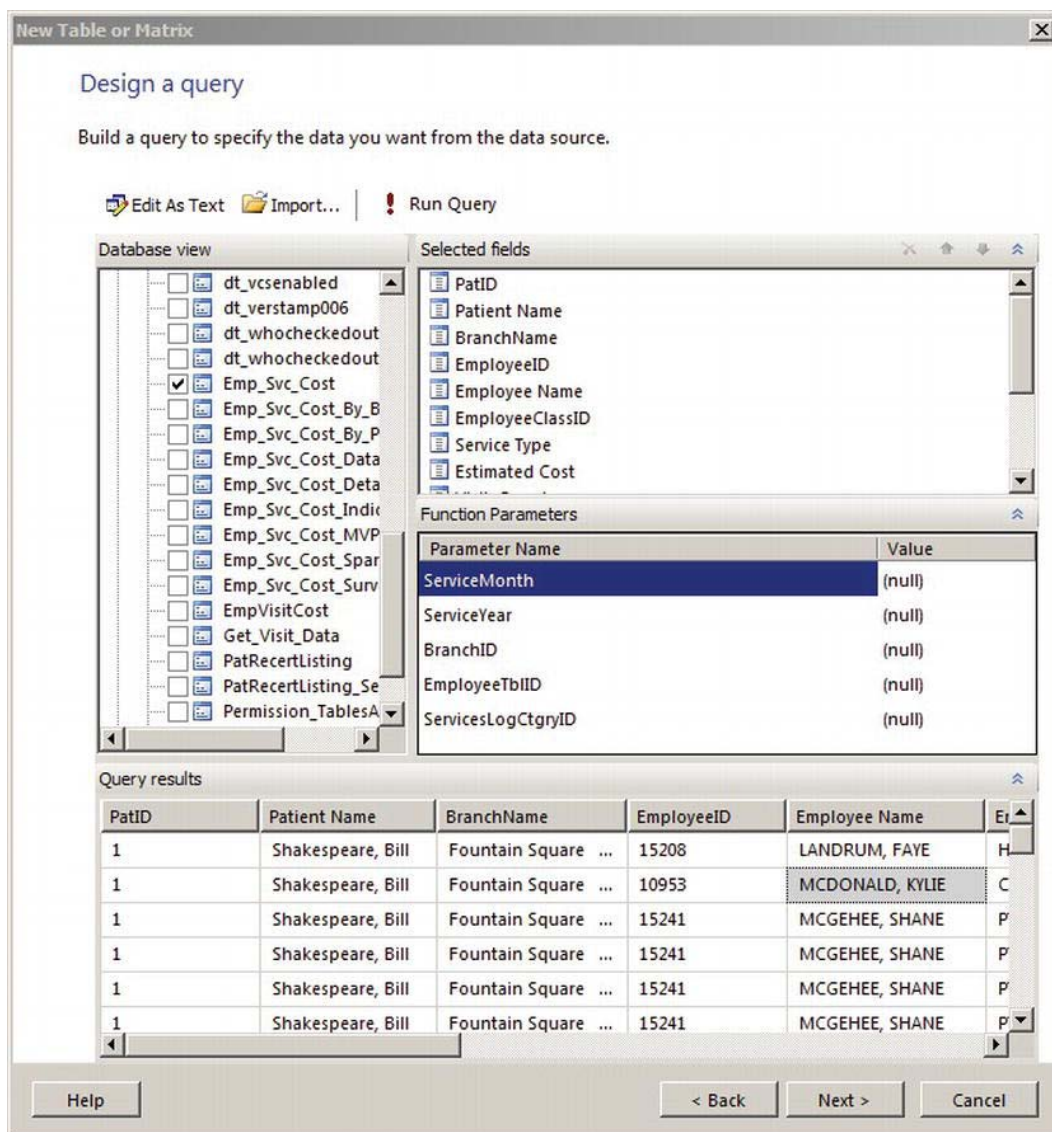


Figure 13-47. Adding the *Emp_Svc_Cost* stored procedure to query designer

After seeing the results, click Next to progress to the Arranged fields screen of the New Table or Matrix wizard. Here is where you layout your report with the appropriate row groups, column groups and values. Use the following steps to produce the *Emp_Svc_Cost* report using the wizard:

1. Drag the BranchName and Service_Type fields over to the Row groups area.

- 2. Next, Drag Visit_Count and Estimated_Cost over to the values area as seen in Figure 13-48.

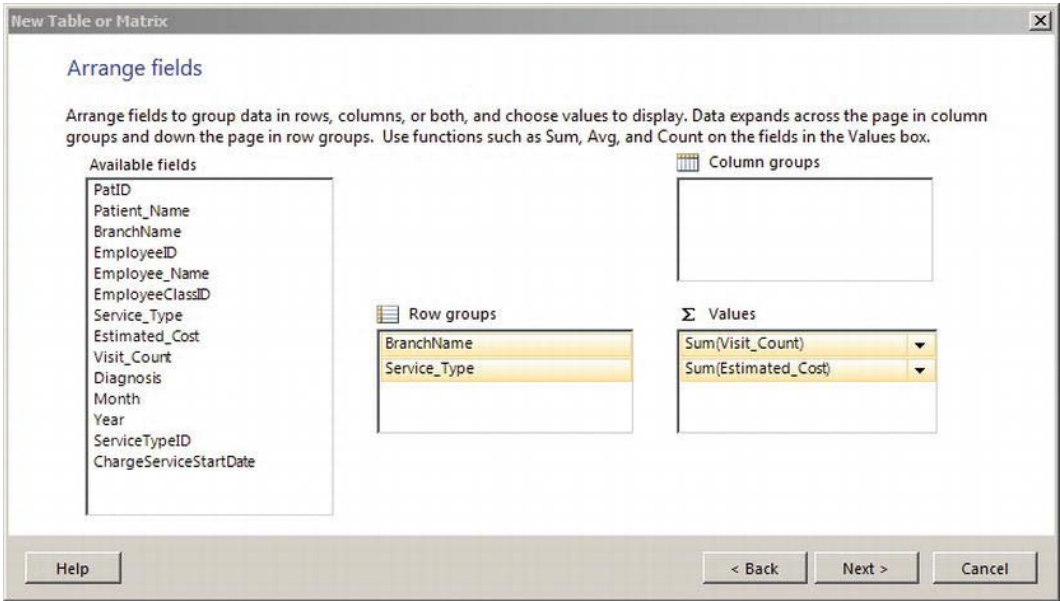


Figure 13-48. Arrange fields screen

- 3. Click Next to proceed to the Layout selection screen. This screen allows us to select a blocked or stepped look for our report. It also has a feature for auto creating expanding or collapsing drill down capabilities using the Expand/collapse groups checkbox. Leave the default values as shown in Figure 13-49 and click Next.
- 4. Leave the default Ocean style selected and click Finish to generate our report.
- 5. Adjust the columns to a width that prevents the header contents from wrapping to the next line.
- 6. Change the title to Service Type Estimated Cost by Branch.

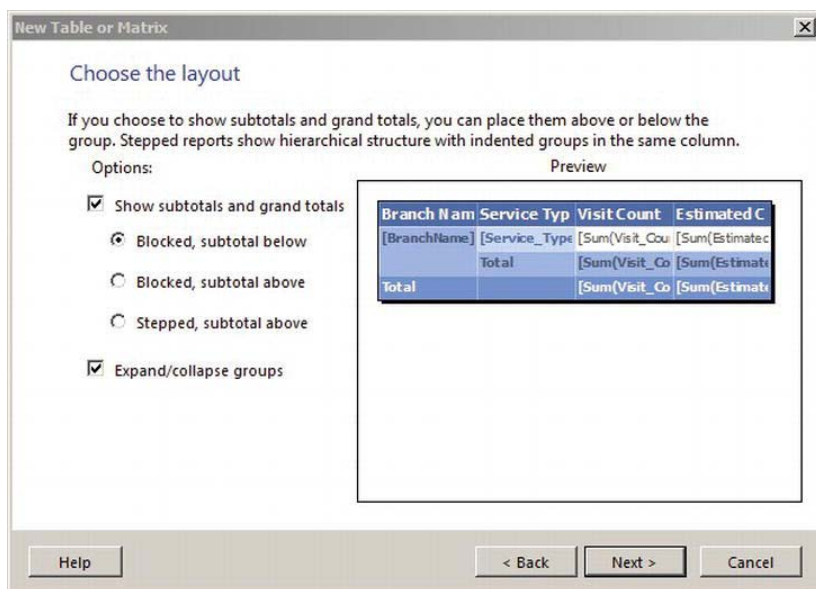


Figure 13-49. Choosing a report layout

With these minor changes performed, we should have a report similar to what is being shown in Figure 13-50.

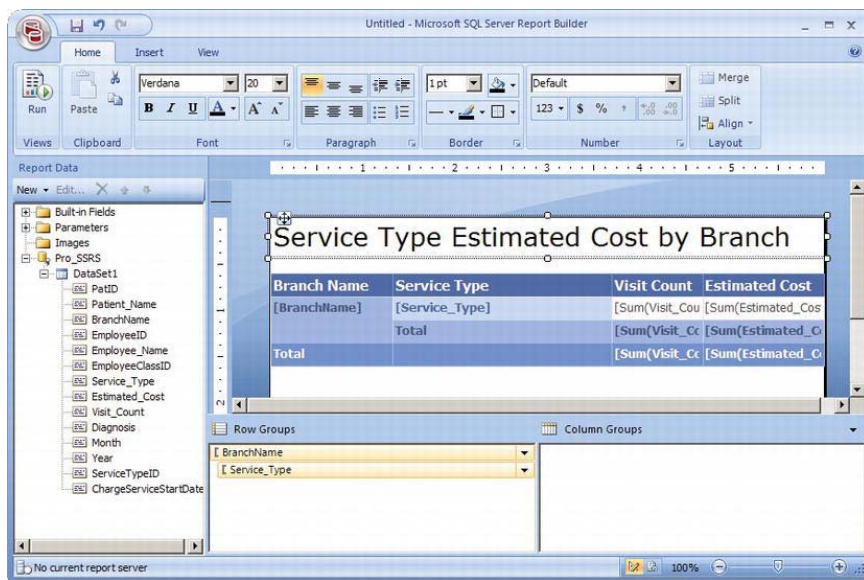


Figure 13-50. Finished report in Report Builder 2.0

Save this report out to the Pro_SSRS folder on the Report Server as in our previous examples and name it Service Type Estimated Cost by Branch Wizard. In the sections that follow, we'll walk through another way to create the same report without using the wizard. We will also show you how to perform format changes to the Report Builder reports.

Creating Reports with Report Builder 2.0

As I just mentioned, we do not have to use the wizard when creating reports. Alternatively, if we didn't want to utilize the New Table or Matrix wizard, we could create a new data source and dataset separately. In this section, we are going to create the same report without using the wizard.

To get started, if you have the Service Type Estimated Cost by Branch report still open, click on the circular ribbon icon on the top left of the screen. This ribbon icon, acts as the File menu option in other Microsoft products. After clicking on the ribbon icon, select New to create a new report. If you do not have Report Builder 2.0 open, navigate to the Report Builder thick client under the Microsoft SQL Server 2008 Report Builder 2.0 from the All Programs menu. If you have been following along, the blank design environment of Report Builder 2.0 should look familiar to you as it is the same as shown in Figure 13-44.

To get started on creating a report from scratch without using the wizard, we will need to create a data source and a data set before getting to the report design. After we have these created, we will be able to add a table to our design pane and then start adding columns and groups. The following four steps will walk us through create the data source and dataset for our report.

1. Click on the New button on the Report Data pane and then select Data Source. If you do not see the Report Data pane, click on the View tab and then select the Report Data checkbox.
2. Name your data source Pro_SSRS and set the Use a shared connection or report model option. Instead of having an embedded data source, we can use a shared one on the server. This helps when deploying reports to different environments. If the reports use shared data sources, if the database server is changed during the lifecycle of the report, the only object that will require updating is the shared data source itself. Conversely, if you always use embedded data sources, each of the reports that utilize the data source would also need to be changes. This makes it harder to manage and as such, one should use shared data sources when possible.
3. Now, select the Browse... button to navigate to the DataSources folder under the Pro_SSRS folder and select the Pro_SSRS data source. Click Open to return back to the Data Source Properties window as shown in Figure 13-51. Click OK to save the Pro_SSRS data source to return back to the Report Builder. After you save the data source, Report Builder adds it the Report Data pane.

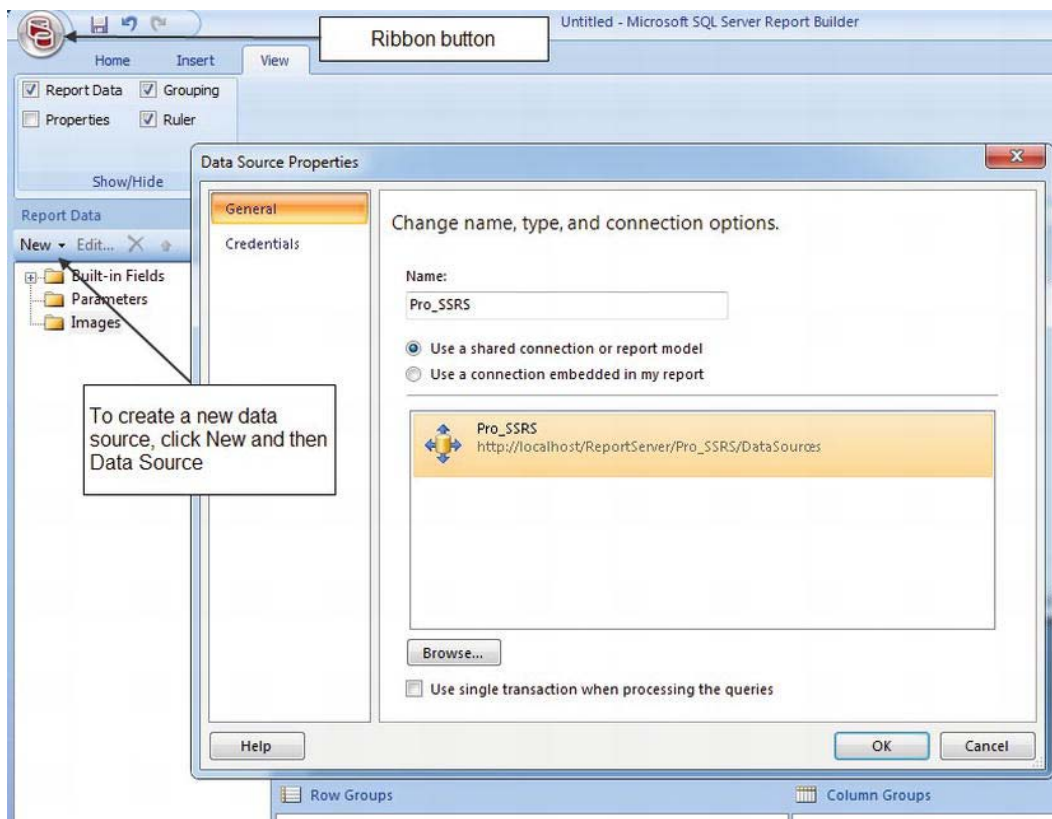


Figure 13-51. Creating a new data source

4. Right-click on the new Pro_SSRS data source in the Report Data pane and select Add Dataset. Name the dataset Emp_Svc_Cost and set the Query type to Stored Procedure. If prompted for login credentials, enter the appropriate login details. Then in the Select or enter stored procedure name drop down, select our Emp_Svc_Cost stored procedure. Figure 13-52 displays the Data Properties window. Click OK to save the Emp_Svc_Cost dataset.

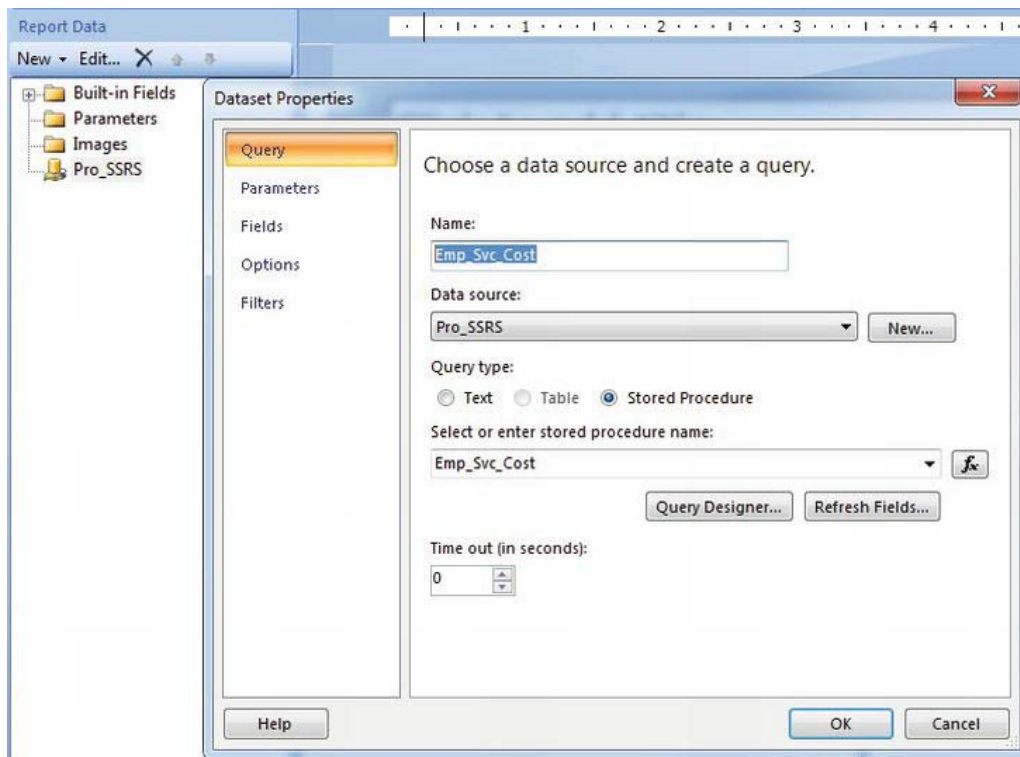


Figure 13-52. *Creating a new dataset*

Once a query has been added and the data source saved, you can see the available fields as well as the parameters that were automatically generated that can now be used in the report. Also, the directional text in the report has changed. As shown in Figure 13-53, you are now instructed on how to add an item to the report body, page header, and/or page footer by clicking the Insert tab or dragging an item to the appropriate area. You will do that next by adding a Matrix control to the report body, a report title to the page header, and the report execution time, a built-in field, to the page footer.

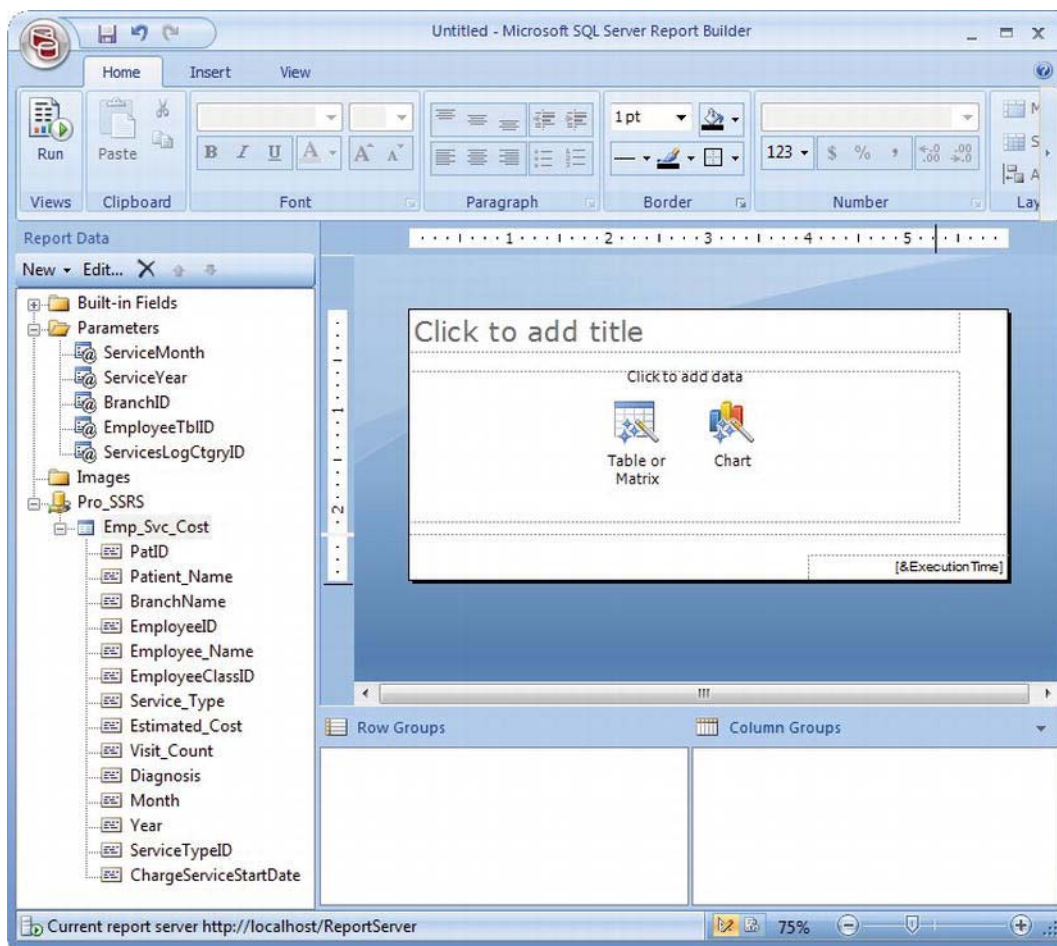


Figure 13-53. Report Builder 2.0 after adding data source and dataset

Now that you have the environment ready to create a report with the newly added data source, click the Insert tab. As you can see in Figure 13-54, all of the report controls are available in four categories at the top: Data Regions, Report Items, Subreports, and Header & Footer. Before we add our own Table, Matrix, or Chart to the design surface, we are going to delete the Data Region Placeholder that contains the two link buttons for the wizards. Just select the DataRegionPlaceholder and hit the Delete key on your keyboard. To add the Matrix data region, simply click the Matrix icon and choose Insert Matrix to add the control to the body of the report. Next, just draw the Matrix onto the design surface to be approximately ½ inch high by 2 inches long.

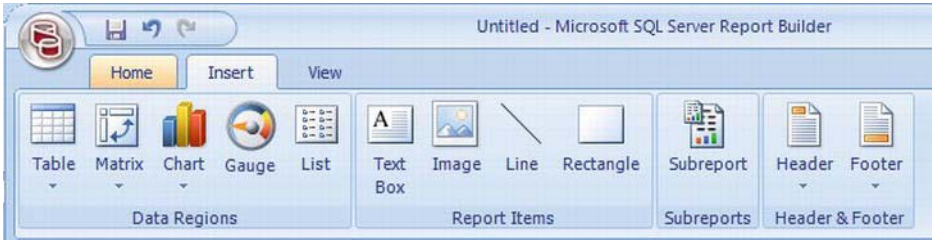


Figure 13-54. Report objects available from the Insert tab

Next, drag the following four fields from the Emp_Svc_Cost dataset that was created with the data source, onto the Matrix control in the report: BranchName, Service_Type, Visit_Visit_Count, and Estimated_Cost. Add BranchName and Service_Type to the Rows area and Visit_Count and Estimated_Cost to the Data area. When we added Visit_Count to the Data area, Report Builder added a header automatically. We don't want it to be labeled with Visit Count, so select the textbox and then hit Delete to remove it. At this point, the report should be similar to the layout depicted in Figure 13-55. Notice that the Sum functions were automatically applied to the fields dragged to the Data area.

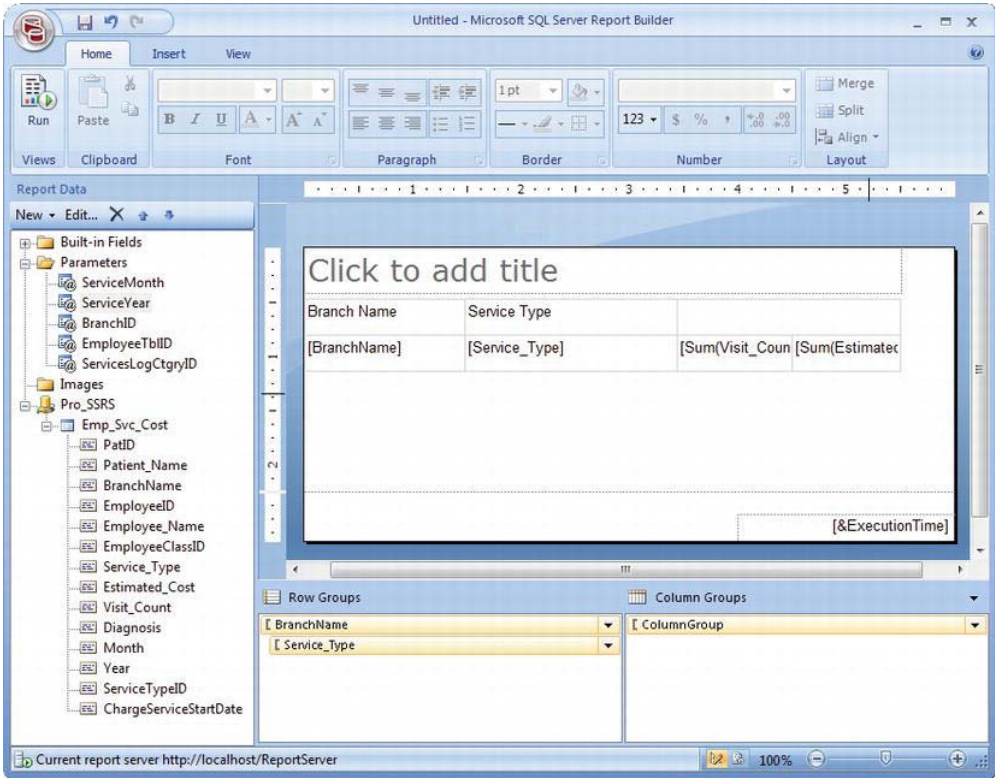


Figure 13-55. Report Matrix with fields added

You will want to resize the cells in the Matrix so that the Service_Type records do not wrap. If you click the View tab, there is a selection for Ruler. Click the Ruler button and size the Service_Type cell to be two inches. Before running the report, let's change all of the parameters to allow NULL values. To set the parameters to accept NULL values, simply double click the parameters and enable the checkbox labeled *Allow null value*. You can see the previewed report taking shape (Figure 13-56) by clicking the Preview button, available on the Home and View tabs.

■ **Note** You may have to set the parameters to allow NULL values in your report if you do not want to have to enter parameter values. By default, if the parameter value is allowed NULL values, the report will generate with all data returned from the execution of the stored procedure.

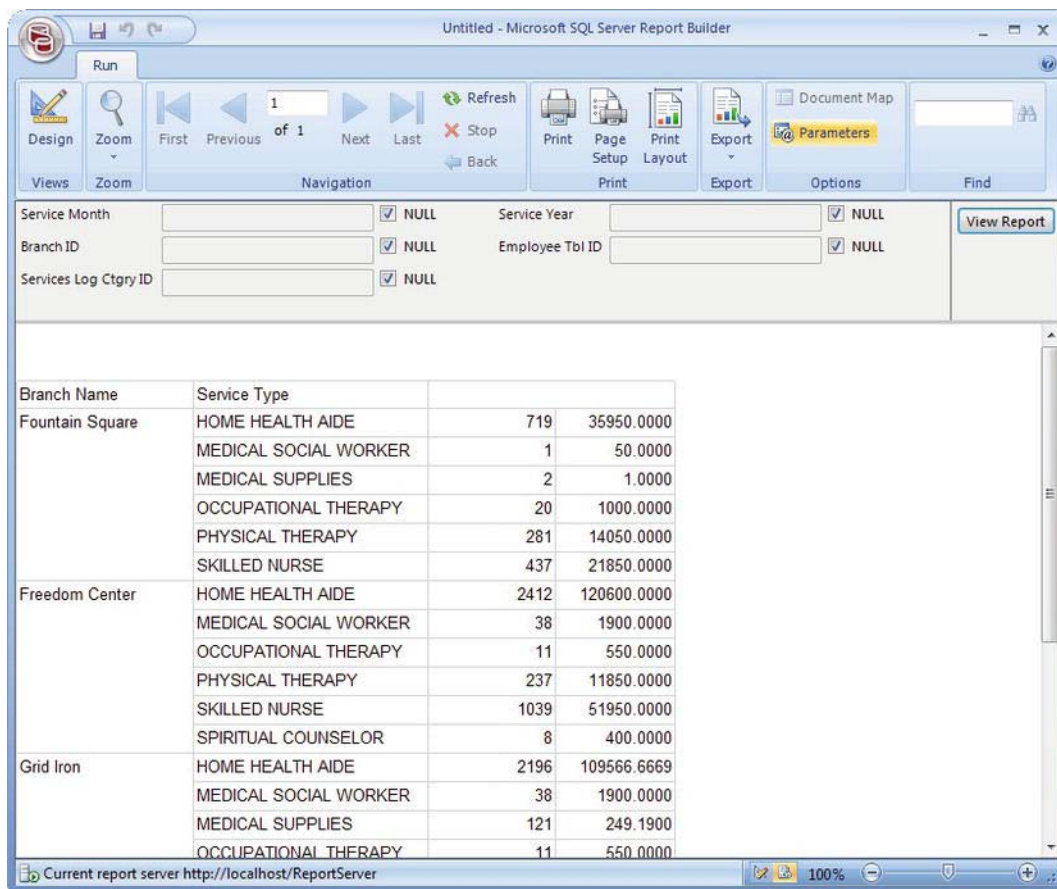


Figure 13-56. Preview of Report Builder 2.0 report

Next, you will format the Estimated_Cost field to display the value as currency. Click back in the Design area, and then follow these steps to apply some formatting:

1. Right-click the SUM(Estimated_Cost) textbox and choose Textbox Properties.
2. Figure 13-57 shows the dialog that presents the textbox properties encompassing many of the formatting and actionable items available, such as interactive sorting, visibility, and hyperlink options. These are all the same as properties that you would set when developing a report in BIDS. Click Number on the left and choose Currency for the format, then choose the option to Use 1000 separator (,) and click OK to save the Textbox Properties.
3. Next, apply the number formatting with the thousands separator to the Visit_Count field by right-clicking it and selecting Textbox Properties. Select the Number tab and then choose Number as the Category. Set the decimal places to 0 and select the option to Use 1000 separator (,) and click OK to save the Textbox Properties.
4. Since the default height of the report has given extra white space, let's cut down on this by dragging the footer bar up closer to our Matrix report item.

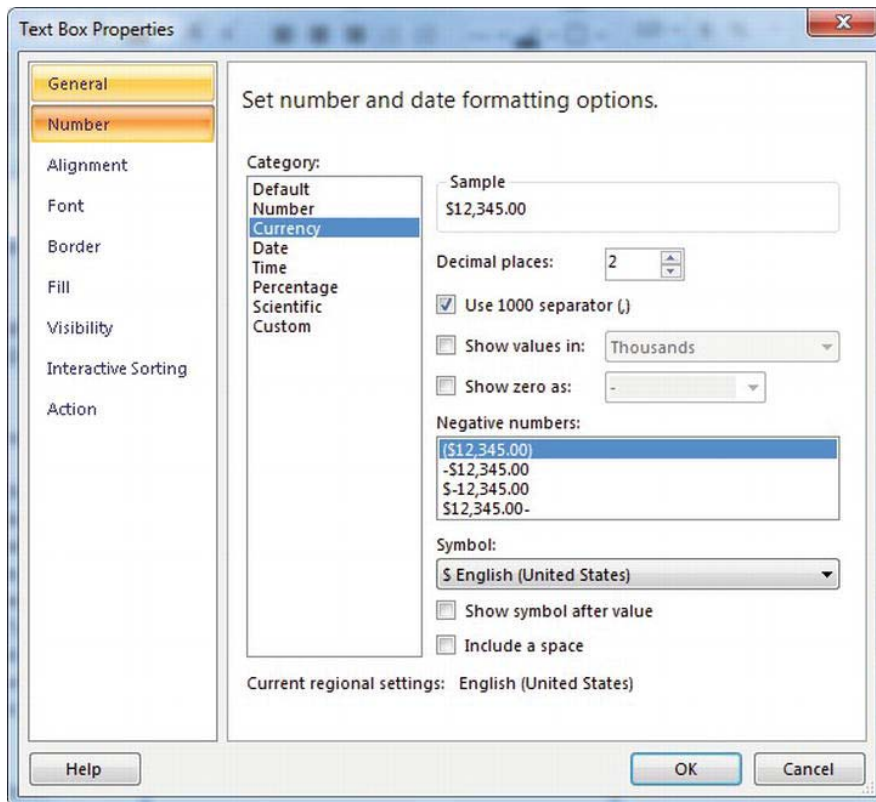


Figure 13-57. Textbox properties for Estimated_Cost

5. Finally, reset the report title by clicking the Click to add title textbox at the top of the report. Set the report title to Service Type Estimated Cost by Branch and change the width of the textbox to be approximately 5 3/4 inches.

When you are done making all of these changes, the report should look like Figure 13-58.

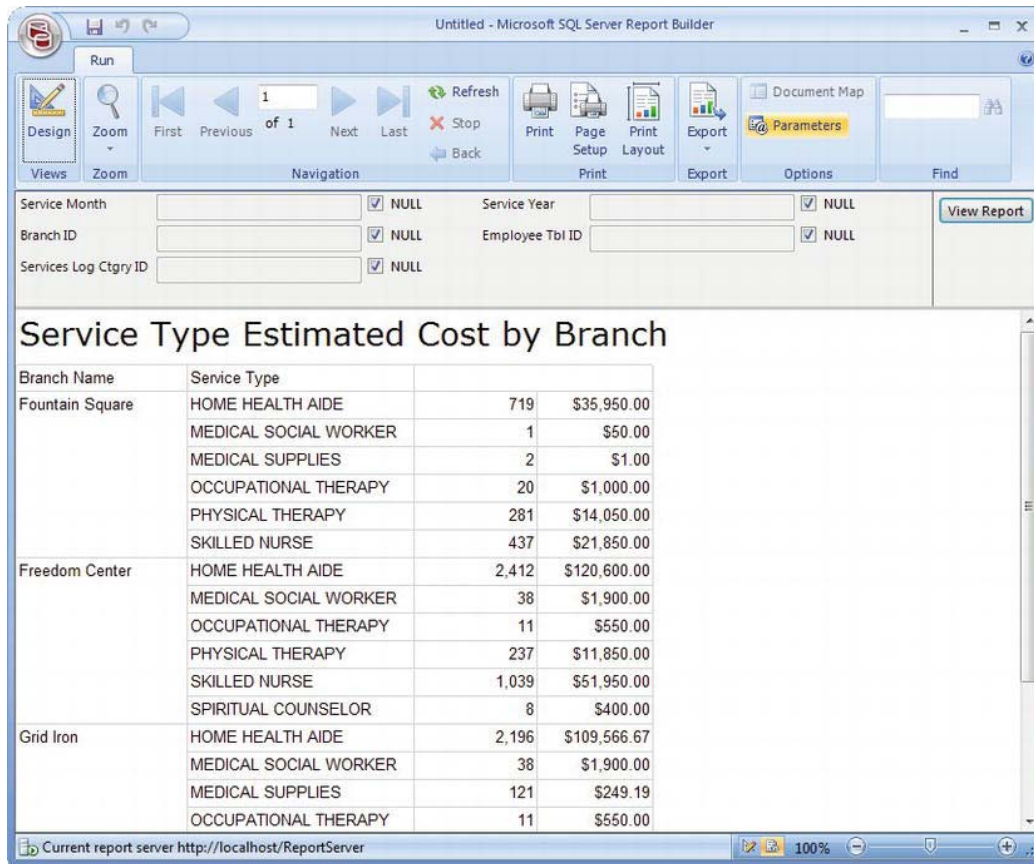


Figure 13-58. Report with title added

Finally, you have probably noticed the [ExecutionTime] in the report footer. This is added by Report Builder 2.0 by default. If you wish to delete it, just select it and hit delete. However, if you want to add more built-in fields to the footer or header sections, just expand the Built-in Fields folder in the Data window to see your options. You will see ExecutionTime, PageNumber Report Folder, Report Name, Report Server URL, Total Pages, User ID, and Language as possible selections. For our example, we are going to add a header to our report and add the User ID to it along with our report title.

1. While in design mode, click on the Insert tab.
2. Click on Header and then Add Header. Resize it to be about 1/2 of an inch in height.

3. Select the title in the body of the report and drag it into the header section
4. With the report title selected, click on the View tab and put a check in the Properties option to show it.
5. Change the FontSize property to 14pt rather than the default 20pt. Resize the header textbox to be slightly larger than the title. This will give us some room in our header to show one more field.
6. Drag the User ID build-in field over to the right of the report title. This will create a textbox with the User ID field value. You can right align this new textbox make it italicized and add bold formatting.

You can easily add more formatting, or report beautification as I like to call it, but at this point, the report should look like Figure 13-59 when previewed.

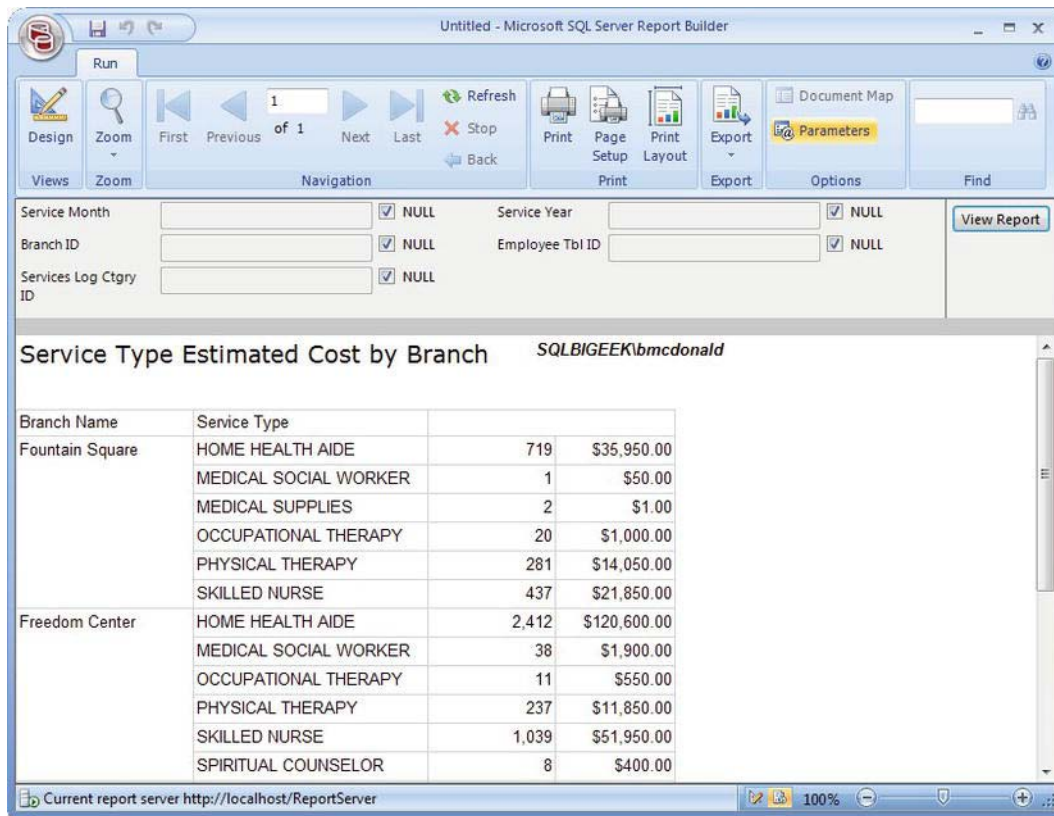


Figure 13-59. Preview of report with execution time and title

For a final touch, you will add totals to the report. This is easily accomplished by clicking back in the design area and selecting both of the cells in the Data region, Visit_Count, and Estimated_Cost. You can select both at once by selecting the first cell and, while holding down the Shift key (or CTRL key),

selecting the second cell. Once you have selected it, right-click and then choose Add Totals. The total fields are automatically added for you. You can emphasize the totals by making the two new total cells bold. When complete, the report will appear as it does in Figure 13-60.

As you have seen, the new Report Builder 2.0 has many of the features of the full-blown design environment in BIDS with only minor exceptions. One exception that stands out is the inability to check a report directly into a version control application such as Team Foundation Server.. However, for its intention, the Report Builder 2.0 does serve the purpose of putting a design environment in the hands of people who can create reports beyond the constraint of report models, but who may not be VB.NET developers per se. Making this valuable design tool available as part of SSRS was one more step forward in bringing SSRS to the masses. In the next section, we will cover the enhancements made in the Report Builder 3.0 application as well as walk through an example using the latest release of Microsoft's Reporting Services ad-hoc tool.

Service Type Estimated Cost by Branch

SQLBIGEEK\bmcdonald

Branch Name	Service Type		
Fountain Square	HOME HEALTH AIDE	719	\$35,950.00
	MEDICAL SOCIAL WORKER	1	\$50.00
	MEDICAL SUPPLIES	2	\$1.00
	OCCUPATIONAL THERAPY	20	\$1,000.00
	PHYSICAL THERAPY	281	\$14,050.00
	SKILLED NURSE	437	\$21,850.00
	Total	1,460	\$72,901.00
Freedom Center	HOME HEALTH AIDE	2,412	\$120,600.00
	MEDICAL SOCIAL WORKER	38	\$1,900.00
	OCCUPATIONAL THERAPY	11	\$550.00
	PHYSICAL THERAPY	237	\$11,850.00
	SKILLED NURSE	1,039	\$51,950.00
	SPIRITUAL COUNSELOR	8	\$400.00
	Total	3,745	\$187,250.00
Grid Iron	HOME HEALTH AIDE	2,196	\$109,566.67
	MEDICAL SOCIAL WORKER	38	\$1,900.00
	MEDICAL SUPPLIES	121	\$249.19
	OCCUPATIONAL THERAPY	11	\$550.00
	PHYSICAL THERAPY	603	\$30,100.00
	SKILLED NURSE	1,473	\$73,650.00
	SPEECH THERAPY	10	\$500.00
	SPIRITUAL COUNSELOR	30	\$1,500.00
	Total	4,482	\$218,015.86

4/21/2012 3:02:05 AM

Current report server http://localhost/ReportServer

Figure 13-60. Report with final formatting

Creating Reports with Report Builder 3.0

When we heard that SQL Server 2008 R2 was coming to the market with all of the new visual enhancements, there was talk about another Report Builder. After all of the great additions made with Report Builder 2.0, we could not wait to see what ad hoc goodies the next release would bring to our end users.

When SQL Server 2008 R2 came to the market, along came Report Builder 3.0 and with it came some additional data sources, data visualization report items, and cached data sets when previewing reports to name a few. High-level details about each of the enhancements are:

- **Report Parts:** Report Parts allow report authors to create and deploy reports that can be re-used by other reports. For example, if a report developer created a chart for Employee Service Cost by Branch report with a total cost chart, another report developer or end user could use that report as a Part of their report.
- **Data Sources:** With the coming of age of SQL Azure, Microsoft has created a Microsoft SQL Azure data source. They have also added the SharePoint List Data Extension to connect to SharePoint lists and Microsoft SQL Server Parallel Data Warehouse.
- **Report Items:** As discussed in Chapter 5 using the Report Designer in SQL Server Data Tools, new report visualization items have been added to the Report Builder. The new report items are Map, Data Bar, Sparkline, and the Indicator. You can find these on the Insert tab of Report Builder.
- **Dataset Caching:** With end users creating ad hoc reports, the potential for many long running report requests is pretty high. As such, to prevent multiple round trips when no dataset related changes are made, Microsoft has added dataset caching to the matrix of new features. This makes the end users experience better when creating ad hoc reports, which could equate to more acceptance by end users.

Now that you've been introduced to Report Builder 3.0 and some of the new features and enhancements, let's get started on some examples. In this example, we are going to build a report that has the number of visits and the estimated cost of service for each patients State and by the State in which the patient was seen at one of our branches. It will be very similar to the report that we have created with the exception that we will also be using a Map control. We will also have a drill through report from our Map report to a more detailed tabular report. We have learned about drill through reports earlier in the text, but we never performed a drill through using Report Builder. The drill through report demonstrated in this section is available in the Pro_SSRS project in the Source Code/Download area for the book on the Apress Web site (www.apress.com). This report is called `Emp_Svc_Cost_By_Patient_State_RB3_Drill`.

Before we get started on Map report, open up Report Builder 3.0 from the Report Manager. Once Report Builder is loaded, click Open in the Getting Started window. Navigate to the location where you have the Pro_SSRS solution stored on your hard drive and select the report named `Emp_Svc_Cost_By_Patient_State_RB3_Drill`. Click Open to open the report in Report Builder 3.0. If you click Run, you should have similar results to what is shown in Figure 13-61.

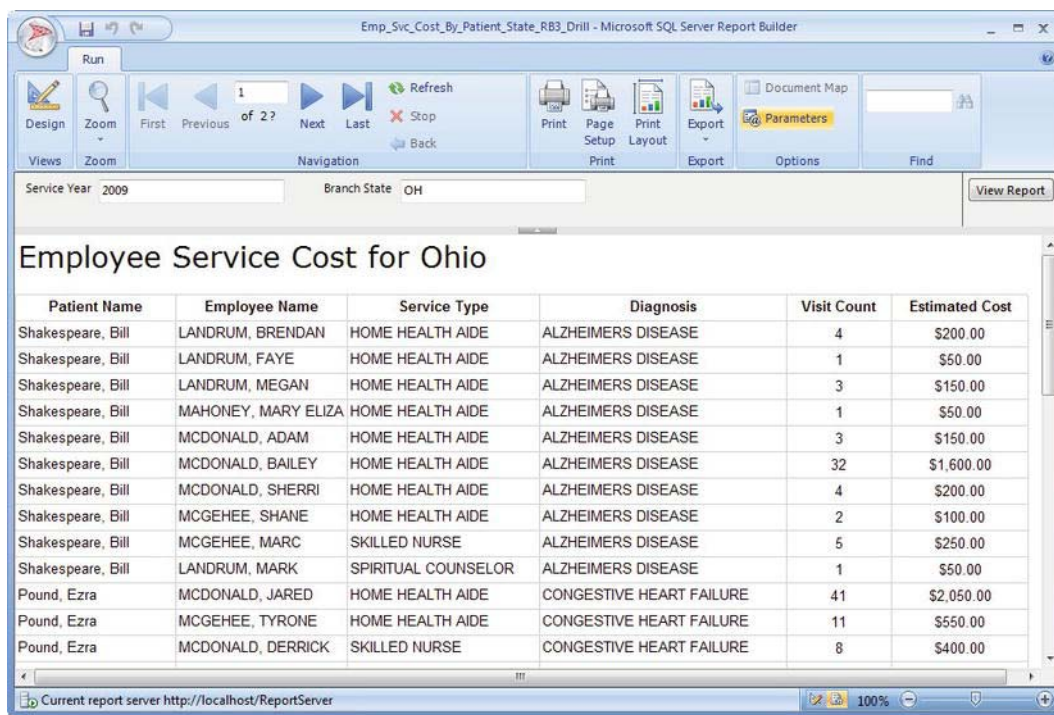


Figure 13-61. Report Builder 3.0 drill through report

After you preview the report, return to the designer and save the report out to the Pro_SSRS folder, as we have throughout this chapter. Saving the report in this path will allow us to drill through to this report when we have our Map report created. With the report saved to the Report Server, click on the Ribbon button and click New. As stated previously, our next report is going to contain aggregated visit and estimated cost for each state that a patient is from as well as which state the patient was actually seen at. Figure 13-62 shows an example of the report while in design mode.

■ **Note** Again, we have been using sample data that has been changed to provide result sets that represent real world scenarios. With our fictitious patients coming from around the country, it is not very likely that they would travel from states like California, to be seen in Ohio. Nonetheless, the example in the following section could be used for many applications that have a requirement to provide maps along with quantifiable data.

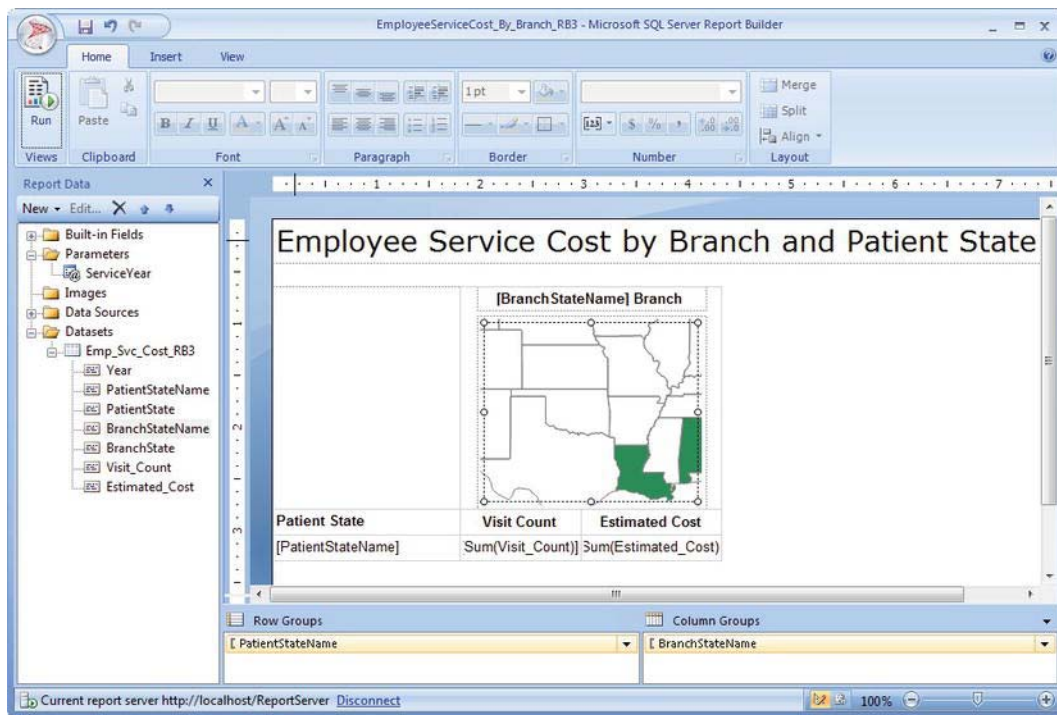


Figure 13-62. *Employee Service Cost by branch and patient state in design mode*

1. Right-click on the Data Sources folder and select Add Data Source. Name the data source Pro_SSRS, select Use a shared connection or report model and click the Browse button to navigate to the Pro_SSRS data source under the Pro_SSRS/DataSources folder. Click Open and then OK to save the data source.
2. Right-click on the Datasets folder and select Add Dataset. Name the data set Emp_Svc_Cost_RB3 and select Use a dataset embedded in my report. Select the Pro_SSRS as the data source to use and select the Query type radio button to Stored Procedure. Select the Emp_Svc_Cost_By_Patient_State_RB3 stored procedure and click OK to save your dataset.
3. Next, change the report title to Employee Service Cost by Branch and Patient State. Resize the title textbox to fit the new title without wrapping to the next line.
4. Right click on the report design surface, select Insert and then Matrix. This adds a new Matrix report object to the report design pane. Drag the Matrix toward the top left near the title.
5. Drag the Visit_Count field into the Data field and then drag the Estimated_Cost field to the right of Visit_Count.

6. Delete the Visit_Count label that Report Builder put in the Matrix Column field when we dragged Visit_Count over onto our Matrix.
7. Drag the PatientStateName field into the Rows field of our Matrix to create a row grouping for each of the Patients States. Delete the Patient State Name label that was placed in the top left corner of the Matrix as we did for the Visit Count label in step 6. Using the Border Lines formatting option, set the lines around the empty top left corner field to be None.
8. Insert a Rectangle report object in the Columns field by right-clicking in the columns field, selecting Insert and then Rectangle. This is where we are going to put the BranchStateName and a Map for the State. So that we have some room for our BranchStateName and a Map, make the height of this row to be about 2 inches as shown in Figure 13-62 shown previously.
9. Right-click inside of this new Rectangle, select Insert, and then Textbox. Move the Textbox to the top of the Rectangle and set its width to be 2.5 inches. Right-click the textbox and select Expression. Set the expression so that it reads Fields!BranchStateName.Value. Click OK to save the expression. Center the alignment of the text and make it Bold. Then enter a space after the [BranchStateName] and then enter Branch. The textbox should now read [BranchStateName] Branch.
10. Select a blank section in the Rectangle just below the BranchStateName textbox that we just created. Using the Home tab on the Ribbon, change the color of the lines surrounding the rectangle to be Silver and then click on the border to place lines around the entire rectangle. At this point, our report should look similar to Figure 13-63.

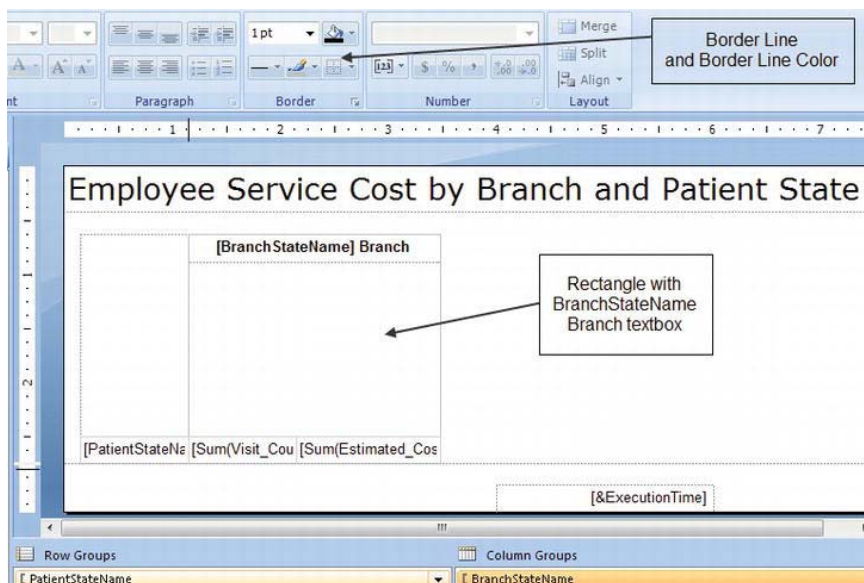


Figure 13-63. *Employee Service Cost by branch and patient state*

- 11. Format the [Sum(Visit_Count)] field to be Number with no decimal places by selecting the field and then selecting the Properties on the View menu. Navigate to the Format property under the Number category. Set the format to be N0 (that is N and zero). Center the field.
- 12. Format the [Sum(Estimated_Cost)] field to be Currency with two decimal places by setting its Format property to C2.
- 13. Add a row above the PatientStateName, Visit_Count and Estimated_Cost row by right-clicking on the PatientStateName field, selecting Insert Row and then Outside Group – Above. Now that we have these extra fields, let's put labels in them to represent the columns that are below them. Set the first empty textbox to read Patient State, the second to Visit Count, and the last one to Estimated Cost. Make each of the fields Bold and widen the Patient State column to be about 1.75 inches wide. Click Run to see the fruits of our labor up to this point. Set the Service Year parameter to 2009 and click View Report. Figure 13-64 shows what the report should look like if you have been following along.

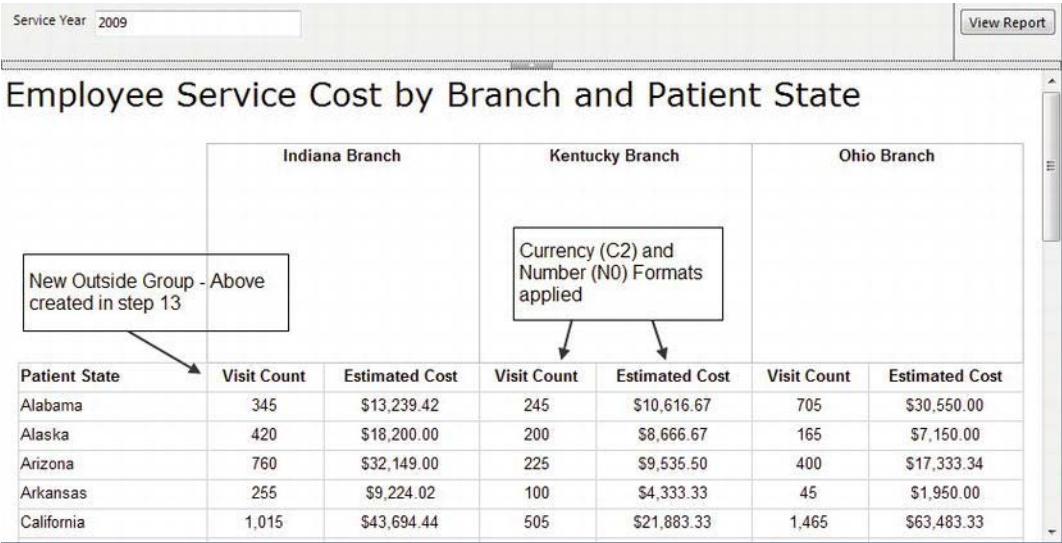


Figure 13-64. Previewing Employee Service Cost by branch and patient state after outside group and formatting

- 14. As you can see, we have left some space below the branch State names textbox and the new group. Now we are going to add a Map to that empty space. Click on the Design button to return to the Report Builder designer.
- 15. Right-click in the empty space below the BranchStateName textbox, select Insert and choose to insert a new Map report object. Report Builder will make the rectangle much larger, but we will shrink it down after we get the Map just the way we want it.

16. To get started, we are going to remove the extra items that are added by. Right-click on the Map and remove the checkbox from Show Meridians. Do it again to remove the checkbox from Show Parallels. Perform the same action for the Color Scale (bottom left) and the Distance Scale (bottom right). Right-click on the legend and then select Delete Legend. Perform the same action for the Map Title. At this point, the Map looks like a giant grey rectangle right? I would say that we are all done, but that would be a joke. In actuality, now we are ready to work some mapping magic.
17. Right-click the map, select Map, and then Add Layer. Leave the default Map Gallery selected, choose the USA by State Inset, and then click Next. We will set the zoom to be dynamic later on, so for now, click Next on the Choose spatial data and map view options screen to progress through the New Map Layer wizard. Choose the middle option, Color Analytical Map and click Next. We could create a new dataset or browse for another one, but we already have one that we can use. Select our Emp_Svc_Cost_RB3 dataset that already exists in our report and hit Next. In the Specify the match fields for spatial and analytical data screen, put a checkbox in STATENAME, and then select BranchStateName as the Analytical Dataset Field to map to Click Next. Set the Theme to Generic, Field to Visualize to BranchStateName, and click Finish to complete the map layer wizard.
18. Click on one of the States to bring up the Map Layers box and click on the down arrow next to the image of the eye. Select Polygon Color Rule. For this example, we are going to choose the option to *Visualize data by using custom colors*. Next, click the Add custom colors button and set the color to be Sea Green. Click OK to save the Map Color Rules Properties.
19. Resize the map to be the same width of the BranchStateName textbox and approximately 2.5 inches in height. Resize the width of the column that contains the map to be about 2 inches wide. Set the height of the row containing the map to be around 2.5 inches.
20. Right-click on the map and select Viewport Properties. Navigate to the Fill tab and set the Fill style to be Solid. Click on the Border tab and set the Line style to None. Click on the Shadow tab and set the Shadow offset to be 0pt. Now to have the map dynamically zoom into the State, click on the Center and Zoom tab. Change the setting to Center map to show all data-bound map elements and set the zoom level to be 1000. Click OK to save the Viewport Properties.
21. Expand the Parameters folder in the Report Data pane. Next we're going to add a few year values for our ServiceYear parameter. Double-click on the ServiceYear parameter to open its properties window. Click on the Available values and select Specify values. Click the Add button three times to hold three different values. For the first value, enter 2009 as the Value. Enter 2010 for the next one and 2011 for the last one. Click on the Default Values tab and choose the option to Specify values again. Click the Add button and set its value to be 2009. Click OK to save the Report Parameter Properties. The report should closely resemble what is shown in Figure 13-65.

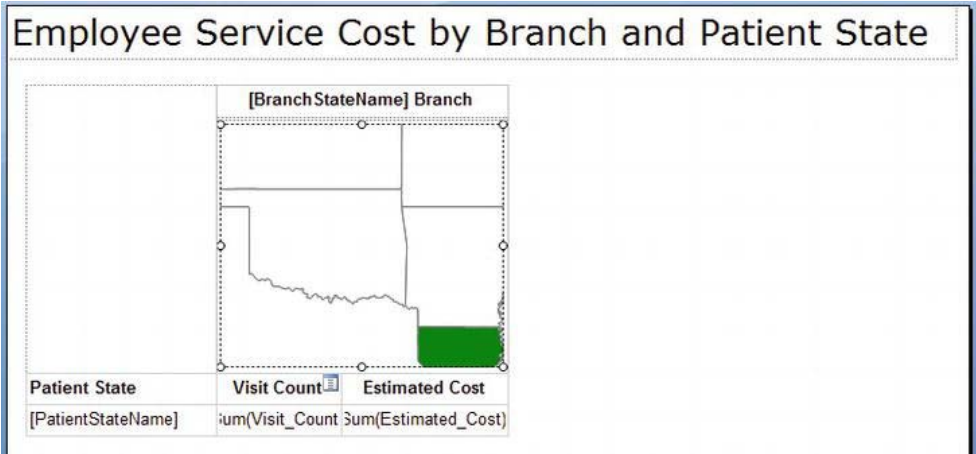


Figure 13-65. Employee Service Cost by branch and patient state after outside group and formatting

22. We are almost complete! One final requirement is to have the ability to drill through from this report to our detailed report named Emp_Svc_Cost_By_Patient_State_RB3_Drill that we deployed earlier. To do this, we are going to create an action on the map. To do this, click on a State again to bring up the Map Layers window. Click on the down arrow as you did earlier, select Polygon Properties and then click on the Action tab. Choose the option to Go to report and click the Browse button to choose the report to run when clicking the map. Navigate to the Pro_SSRS folder and select the Emp_Svc_Cost_By_Patient_State_RB3_Drill report. Click Open to return to the Polygon Properties window. Since the drill through report requires two parameters, we need to specify what parameters will pass through it. Click the Add button twice and set the Name and Value pairs as shown in Figure 13-66. Click OK to save the Polygon Properties Action settings.

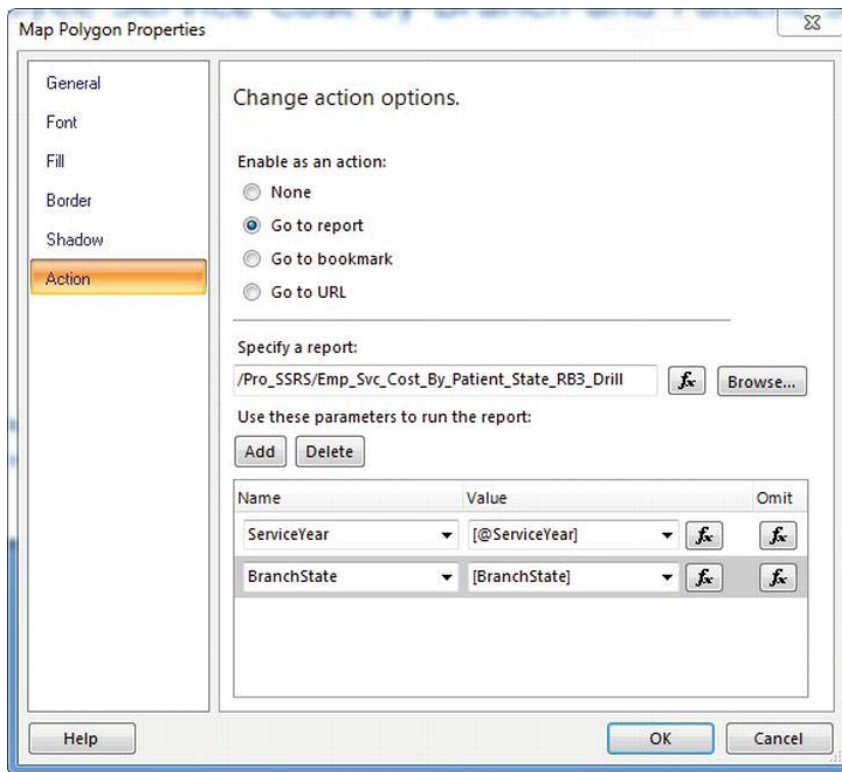


Figure 13-66. Polygon properties action settings

Whew! That was fun! Click the Run button to execute our finished report in action. If you have performed the preceding steps, your finished report should be something close to Figure 13-67.

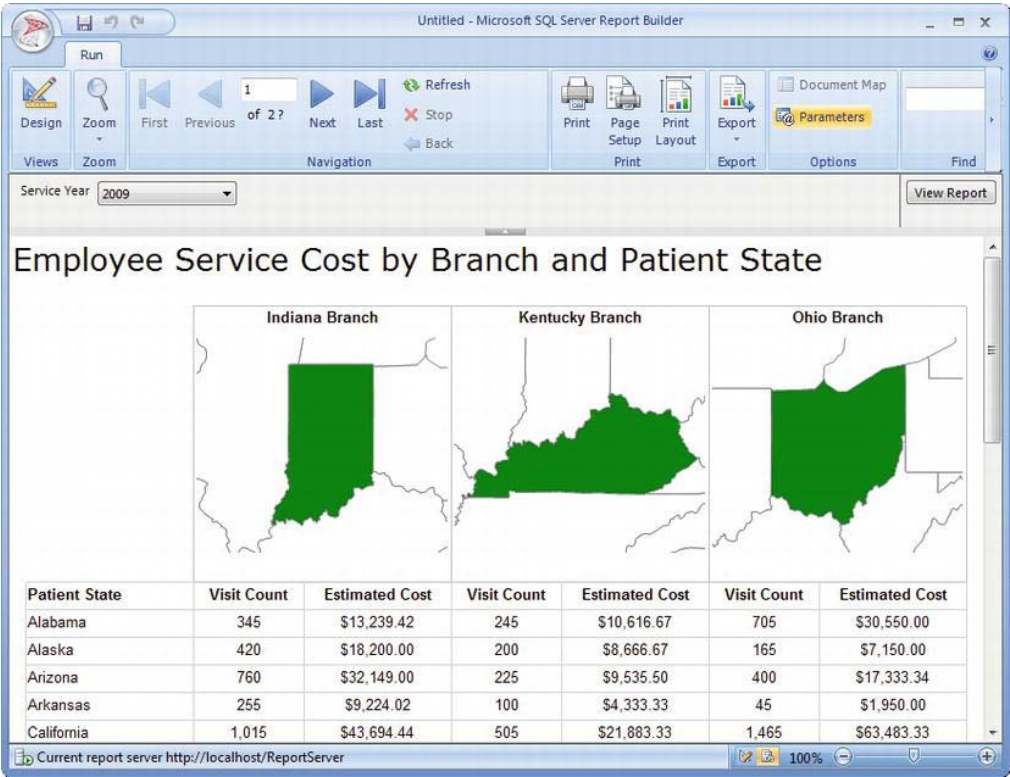


Figure 13-67. Employee Service Cost report by branch and patient state rendered

After running the report, click on one of the branch State maps to drill through to the detail report. If you were to click on Ohio for Service Year 2009, your results would be like those shown in Figure 13-68.

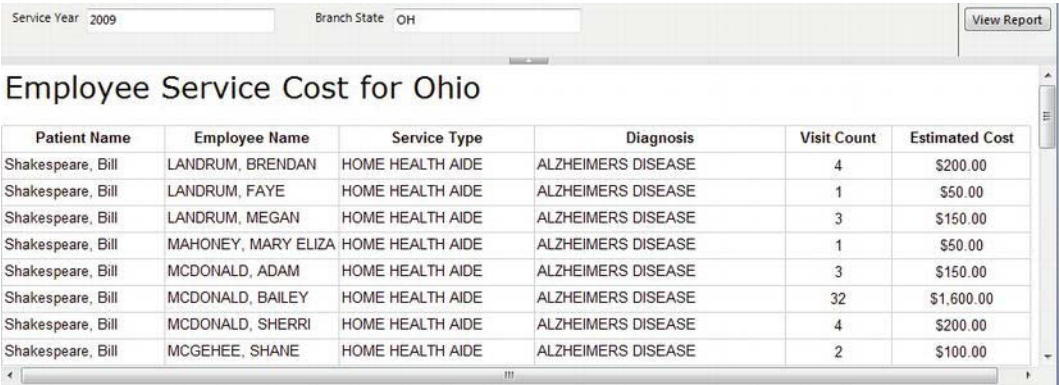


Figure 13-68. Employee Service Cost Detail drill-through report by branch and patient state

Report Parts

There is one more feature that was added with SSRS 2008 R2 that provides some of the true power of ad hoc reporting. As we briefly touched on previously, Report Parts can be created and deployed to the reporting server with the sole purpose of being used by other report developers or ad hoc report authors. Report Parts not only assist in keeping consistency through reusability, but they can also improve report authoring efficiencies. What is also amazing with Report Parts is that when reports consume report parts, if a change is made to the report part, then each report that is using it is notified of the change. The report author can choose to use the new version or keep the existing version that the report is using. This is incredible, I know, and it is a very well received addition to the Report Builder tool.

Before we get into an example, just note that not all report items can be published as report parts. Most of the report items found in the toolbox can be used as a report part, but not all. A list of the report items that can be used are:

- Tablix – Table, List and Matrix
- Charts
- Gauges
- Maps
- Rectangles
- Images
- Parameters

We are going to deploy an existing report to the Report Server as a Report Part. We could do this using the Report Designer or Report Builder, but since we are showing the capabilities of Report Builder 3.0, we will show you how to open up an existing report and deploy it from Report Builder. After we have it deployed, we will show you how report authors using Report Builder 3.0 can “consume” or re-use parts of reports that have been deployed to the Report Server. Open up the Report Builder and let's get started. Here's what to do:

1. Once you have Report Builder started, click on the Report Builder icon in the top left corner and click Open.
2. Navigate to the location that you have extracted the Pro_SSRS solution and locate the EmployeeSvcCost_RB3_ReportPart.rdl file. Click Open to load the report in Report Builder 3.0.
3. Click Run to view the report to see it in action. The report shows a pie chart with the estimated cost for each branch as shown in Figure 13-69.

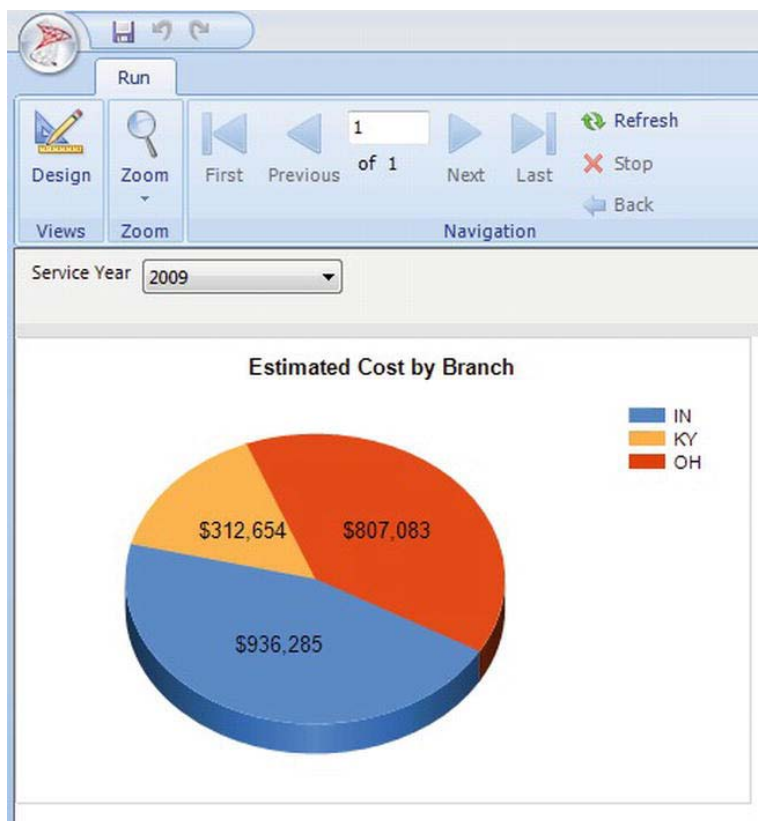


Figure 13-69. EmployeeSucCost_RB3_ReportPart report

4. Click the Design button to return to design mode. Next, click on the Report Builder menu button again and choose Publish Report Parts, as shown in Figure 13-70.

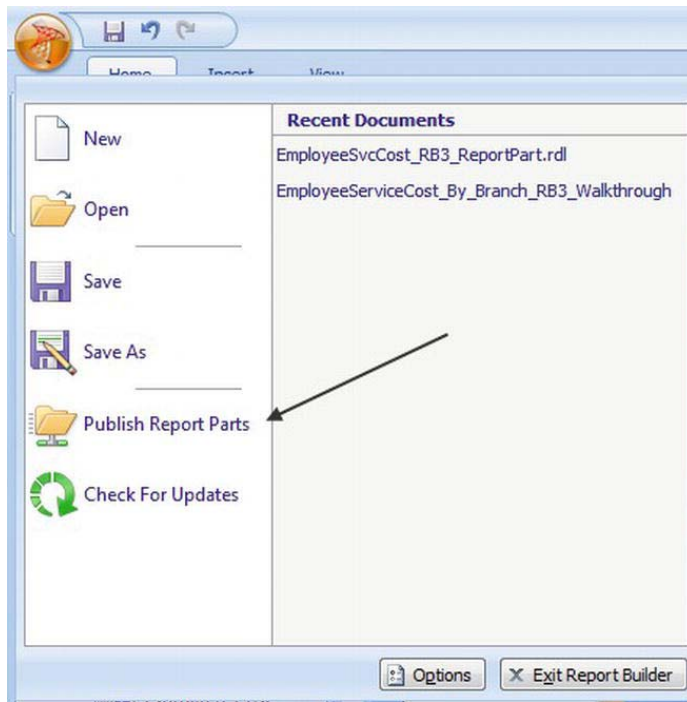


Figure 13-70. Publish Report Parts

5. On the Publish Report Parts screen, we have the options to Publish all report parts with default settings or to Review and modify report parts before publishing. We could choose the former option, but to show you the ability to deploy individual report items, select the latter option: Review and modify report parts before publishing as depicted in Figure 13-71.

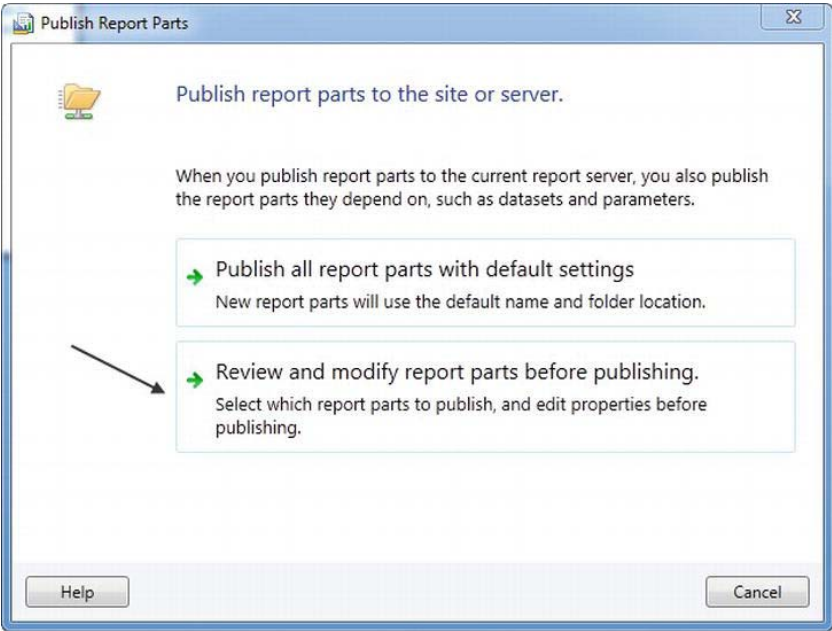


Figure 13-71. Review before publishing

- 6. As can be seen in Figure 13-72, we have the option to deploy two items in our report. The first being the EstimatedCostByBranch_Pie chart and the second one is our ServiceYear report parameter. Since the Chart needs the parameter, ensure that both are selected and click Publish to deploy our chart to the reporting server.

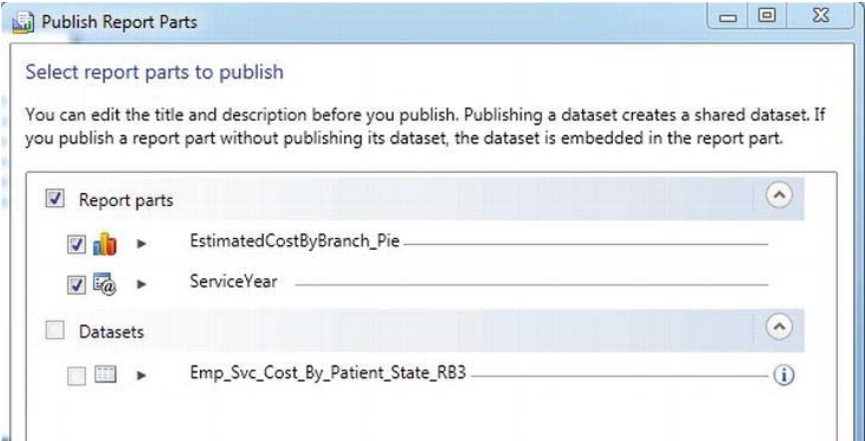


Figure 13-72. Publish selected report parts

7. If permissions are set appropriately (Content Manager) on the reporting server, you should see two green checkmarks next to the report parts that we had selected to deploy. Click Close to return to Report Builder.
8. Now that we have our report parts deployed, now it is time to create a report that consumes the report part. Click on the Report Builder menu button again and choose New and then choose to create a Blank report. After the new report is loaded change the title of the report to “Estimated Cost by Branch”.
9. Click on the Insert tab in the tabstrip and then select Report Parts. You should see the Report Part Gallery window pane become visible on the right side of the screen, as shown in Figure 13-73. From the Report Part Gallery window, you can search for report parts and see things like what report parts are out there as well as who created them. You can even see what the report part looks like when previewing in thumbnail view.

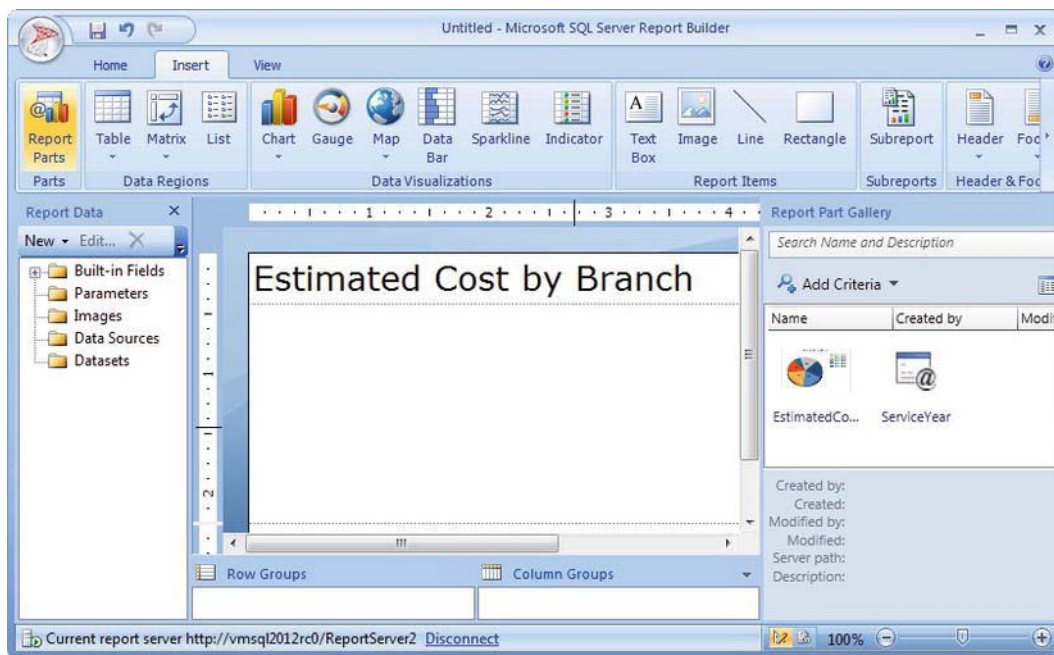


Figure 13-73. Viewing the report part gallery

10. To use a report part in a report, it is as easy as dragging and dropping the report part onto our design surface. With that being said, drag the EstimatedCostByBranch_Pie chart over onto the design surface and set the left side to be aligned with the header. Click on the home tab and then run the report. Executing the report with ServiceYear 2009 can be seen in Figure 13-74.
11. Save the report to the report server as we did in earlier examples with the name of EstimatedCostByBranch_RB3_Report_Using_ReportPart. We will be looking at this report this later after modifying the report part.

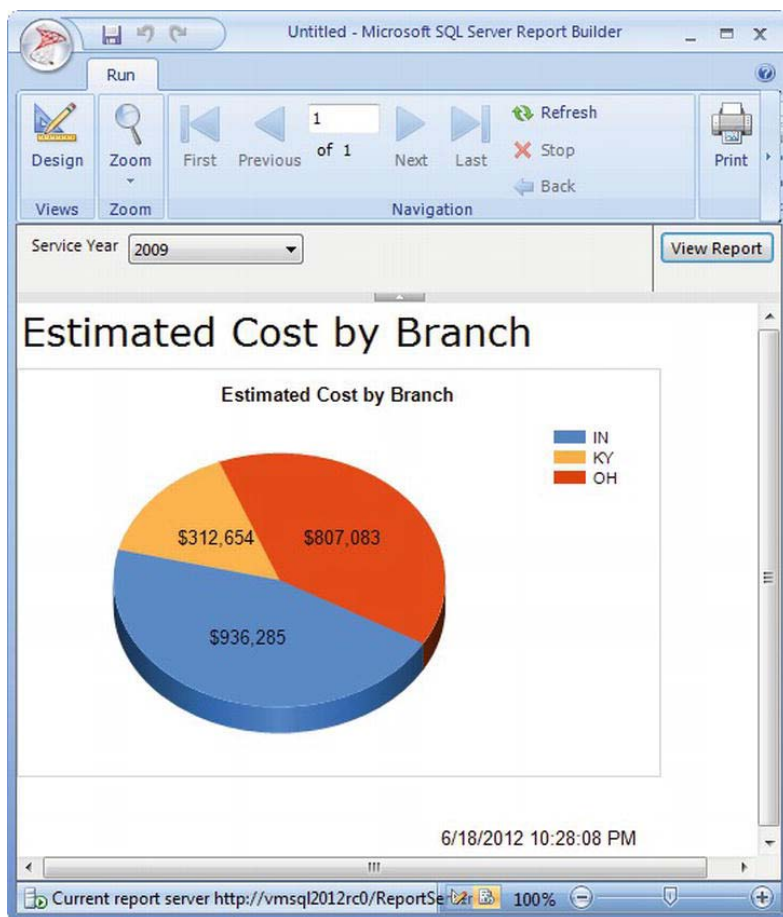


Figure 13-74. Report consuming deployed report part

As you can see, deploying and consuming a report part is a pretty straightforward process. If architected appropriately and some solid report developers, many report parts could be stored on the report server and re-used by report authors to simplify report development.

■ **Note** The report used in this section (“Reports Parts”) is called `EmployeeSvcCost_RB3_ReportPart.rdl` and is included in the `Pro_SSRS` solution that we have been using throughout this book. The `Pro_SSRS` solution can be found as part in the Source Code/Download area for the book on the Apress Web site (www.apress.com).

As you might recall, I mentioned earlier that when changes are made to a report part and it is saved out to the report server, each of the reports that use that report part are then notified of the example. Use the following steps to see this in action:

1. Once you have Report Builder started, click on the Report Builder icon in the top left corner and click Open.
2. Navigate to the location that you have extracted the Pro_SSRS solution and locate the EmployeeSvcCost_RB3_ReportPart.rdl file. Click Open to load the report in Report Builder 3.0. This report is the same report that we opened up earlier. We are now going to make a minor change and then re-publish the report part.
3. Change the font color of the chart title to be Blue and italicized. No major changes, we just want to see what happens when you change a report part. Deploy the report part using the Publish Report Parts as we did earlier from the Report Builder menu button. Only publish the chart this time.
4. Now, let's go and open the EstimatedCostByBranch_RB3_Report_Using_ReportPart that we saved to the report server which has a copy of the original report part. Click the Report Builder menu button and select Open. Navigate to the report server location where you saved the report and open the EstimatedCostByBranch_RB3_Report_Using_ReportPart report. When you open the report, you should see a message box titled "Updated Report Parts" pop up between the tabstrip and report design surface (Figure 13-75). This is our indicator stating that our report part that we consumed has been altered.

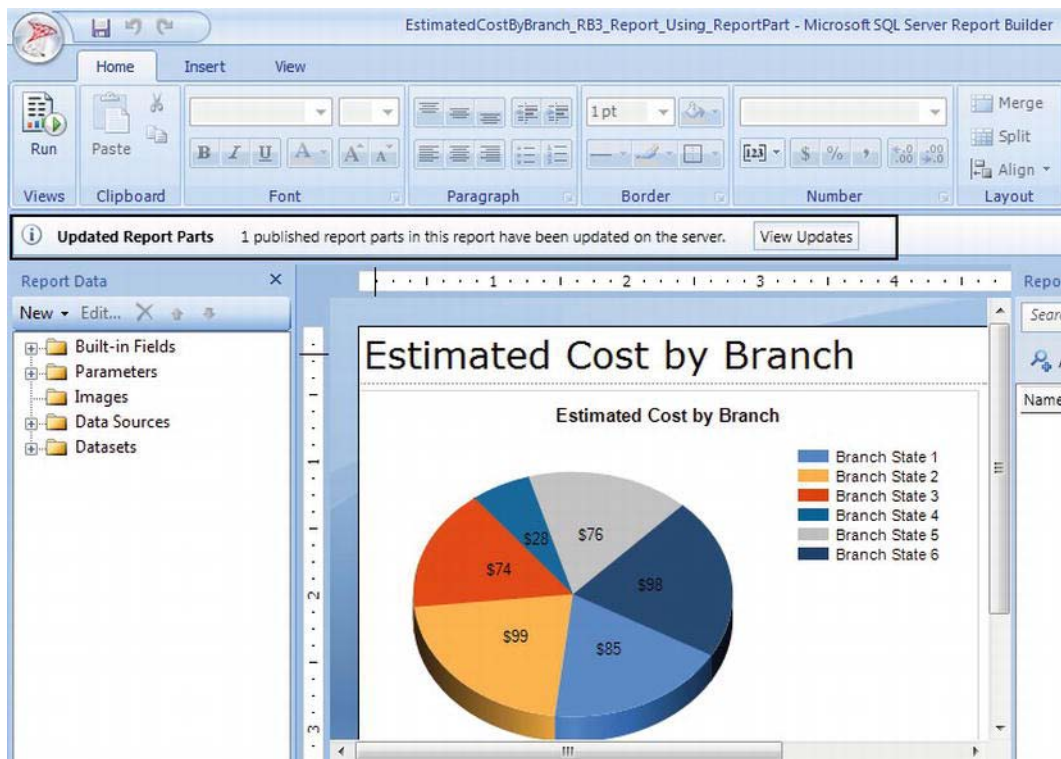


Figure 13-75. Updated report parts message indicator

- Click the View Updates button to see what has changed. When the Update Report Parts screen appears, we should see what has changed as well as a checkbox next to the changed report parts. If we leave the checkbox blank, our report will remain the same as it was and it will not take on these new changes. Conversely, if we place a check in the checkbox, it will update our report with the new report part. We also have the ability to disable the notification when the report part is updated. This essentially tells reporting services that the report doesn't care if the report part ever changes. If you disable this option, you will not be notified of changes. Click the checkbox to update the EstimatedCostByBranch_Pie report part and click Update as shown in Figure 13-76.

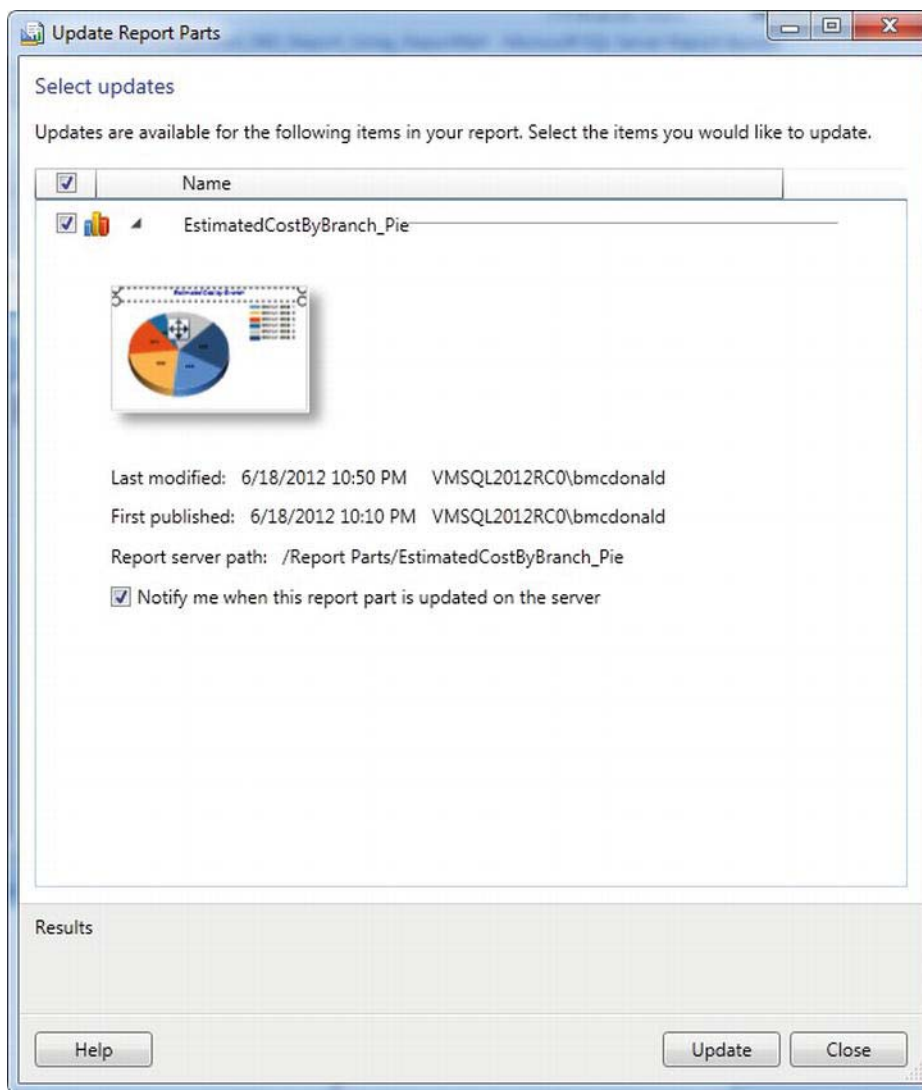


Figure 13-76. Updating reports that consume updated report parts

As you can see by the result of updating our report with the updated report part, we now see a blue, italicized chart title. In my opinion, the Report Part functionality built into Report Builder 3.0 is one of the best additions to ad hoc reporting to date. If you are anything like me, your brain is probably spinning while thinking of new report parts to create for your end users who fall into the report author role—thus, enabling them to quickly and efficiently create ad hoc reports.

Summary

Ad hoc reporting gives users the ability to create their own reports on the fly. It has always been a highly requested feature for any reporting platform and Reporting Services has been no different. As shown in this chapter, Microsoft decided to deliver their first incarnation of ad hoc reporting in the form of Report Builder 1.0. They significantly enhanced the user experience in the Report Builder 2.0 to allow reporting data from relational data and OLAP data sources as well. With the SQL Server 2008 R2 release, came Microsoft's most recent version, Report Builder 3.0. Furthermore, we showed you an amazing feature called Report Parts that allow report authors to reuse existing sections of reports, saving them hours or potentially days in report development time. In this chapter, you learned about some of their respective benefits and some of their limitations. You have probably come away with an answer to the question of whether the Report Builder is a tool that your users will want. If the answer is "Yes," as we hope it will be, then the good news is that these are included as standard features.

Index

■ A

AddNode Method, 237
ATOM, 11
Authentication layer, 10
Average Length of Stay, 436

■ B

BIDS and Visual Studio 2012. *See* Report deployment
Bookmark link, 147–149
Building reports, 125
 chart addition, 171–174
 configuration and group variables, 177–178
 Employee Service Cost report, 125–126
 filter application, 170–171
 final touches, 181–183
 from Scratch, 129–133
 gauge control, 179–181
 interactivity
 bookmark link, 147–149
 document mapping, 137–138
 hyperlink actions, 143–147
 hyperlink formatting and tooltips, 157
 jumping to report, 151–156
 sorting, 141–143
 types, 136–137
 URL link, 149–151
 visibility, 138–140
 multivalued parameters (*see* Multivalued parameters)
 output with formatting, 133–134
 parameters with stored procedures
 BranchID field, 160
 ChargeServiceStartDate field, 163
 datasets addition, 161–162
 Emp_Svc_Cost, 158–159
 EmployeeTblID field, 160
 No parameter values, 161
 NULL values, 159–160
 parameter value queries, 164
 populated parameter selections, 163
 query parameter, 158
 query-assigned values, 160
 Service Year field, 164
 ServiceLogCtgrID parameter, 163
 ServiceMonth parameter, 164
 ServicesLogCtgrID property, 163
 year and month values, 165
Report Wizard
 corporate style, 128
 data source, 126
 generated report, 129
 group and details selections, 128
 Query String area, 127
 tabular/matrix form, 127
subtotals, 134–136
Tablix elements, 175–177
Business intelligence (BI), 1, 6–7
 dashboard, 394–397
 data alerts, 397–399
 report deployment (*see* Report deployment)
 SQL analysis services
 cube wizard, 362
 graphical MDX query builder, 362
 health-care application (*see* Health-care application)
 Microsoft SharePoint 2010, 373
 OLAP solution, 362
 tabular models, 362
 VertiPaq Engine, 362
 stand-alone installation (*see* Stand-alone installation)
Business Intelligence Center, 388–389
Business Intelligence Development Studio (BIDS), 6, 13–14, 40–43
Business Intelligence Semantic Model (BISM), 362

C

- Central Administration, 384
- Chart report creation
 - 3-D Visual Effect box, 444–445
 - Active Patient Census, 445
 - Chart Title, 444
 - data labels, 445
 - data region, 441–442
 - Discharge Date, 443
 - drag and drop data, 442
 - Patient Count formula, 442–443
 - Report Builder 2.0 Wizard
 - data source and dataset, 454–457
 - default Ocean style, 452
 - Emp_Svc_Cost stored procedure, 450–451
 - Estimated_Cost field, 460
 - execution time and title, 462
 - final formatting report, 463
 - finished report, 453
 - Insert tab, 457–458
 - layout selection screen, 452, 453
 - Microsoft's Reporting Services ad-hoc tool, 463
 - New Table/Matrix wizard, 449
 - NULL values, 459
 - Preview button, 459
 - query design, 450
 - Ribbon technology, 447–448
 - Service Type Estimated Cost, 452
 - setting data source and connection string property, 449–450
 - SQL Server 2008 R2/SQL Server 2012, 449
 - SSRS 2008, 447
 - User ID, 461
 - Visit Count, 458
 - Report Builder 3.0 (see Report Builder 3.0)
 - Report Layout tab, 441
 - Report Manager, 446
 - report parts, 473
- Command-line utilities, 12, 14
- Comma-separated values (CSVs), 11
- CreateCatalogItem method, 230, 237, 239, 240
- CreateReportEditSession method, 231
- Custom .NET code
 - class library project, 195–198
 - custom assembly
 - asserting permission, 202
 - code reuse, 195
 - debugging, 207–210

- deploying, 198
 - flexibility, 195
 - language neutrality, 195
 - Named Permission Sets, 200–201
 - productive development environment, 195
 - reference adding, report, 202–207
 - rssrvpolicy.config file, 199
 - security control, 195
 - SSRS 2012, 199
 - task separation, 195
 - troubleshooting, 210
- embedded code, 186, 194
- ExceedMaxCosts function, 187
 - Boolean return value, 190
 - Code member, 189
 - Code.ExceedMaxCost method, 189
 - color property, 190, 191
 - color selection list, 192
 - conditional expression, 190
 - EmployeeServiceCost-NoCode.rdl file, 188
 - expression adding, report, 190
 - expression editor, 192
 - IIF native SSRS function, 190
 - Pro_SSRS.rds file, 188
 - report with the embedded code, 193
 - Visual Studio 2010 BI project creation, 187, 188
- Custom clients, 12, 15

D

- Dashboard, 394–397
- Dashboard-style report objects, 89
 - chart data region, 89
 - Chart Start.rdl, 90, 91
 - Chart.rdl, 95
 - Emp_Svc_Cost stored procedure, 90
 - preview, 93, 94
 - properties, 91, 93
 - RDL output, 94
 - SSRS, 90, 91
 - three fields, 91, 92
 - data bar, 89
 - Chart Data dialog, 111, 112
 - Data Bar Start.rdl, 110
 - Data Bar.rdl, 115
 - object recognition, 110
 - preview, 113, 114

- RDL, 115
 - Series Properties fill options, 112, 113
 - type selection, 110, 111
- gauges, 89
 - Gauge Start.rdl, 100
 - preview mode, 102
 - range, radial, 101
 - RDL output, 103
- image, 89
 - data set properties, 96
 - DocumentImage, 95, 97
 - Image Start.rdl, 95
 - preview report, 99
 - properties, 98
 - RDL elements, 99–100
 - source selection, 96, 97
 - types, 95
- indicator, 89
 - Preview tab, 122
 - properties, 120, 121
 - RDL file, 123
 - type wizard, 120
 - visual, 119
- map object, 89
 - choose map visualization, 105, 106
 - data visualization, 107, 108
 - ESRI shapefiles, 103, 104
 - field specification, 106, 107
 - gallery, 103
 - layer wizard, 105
 - Map.rdl, 110
 - Preview tab, 108, 109
 - RDL output, 109
 - SQL Server Spatial query, 104
- sparkline, 89
 - chart data value, 116
 - control, 115
 - preview, 118
 - RDL, 119
 - Series Properties window, 117
 - Sparkline Start.rdl, 115
 - Sparkline.rdl, 119
- Data alerts, 397–399
- Data processing, 10–11
- Data-driven subscriptions
 - advantage, 303
 - creation, 305–308
 - daily activity report, 303, 304
 - GETDATE function, 304
 - output, 305
 - T-SQL Query, 304

- Data source view creation
 - Add New Data Source, 409
 - additional tabs, 415–416
 - friendly names, 409
 - graphical query designer, 411
 - multiple tables, 409
 - named query, 410, 411, 413
 - New Named Query button, 410
 - patient census query, 411–414
 - Pro_SSRS_2008R2 database, 409
 - report designers, 409
 - report model
 - model generation rules, 418
 - patient census report model, 421–422
 - Pro SSRS 2008R2, 417
 - process completed successfully, 419–420
 - project property, 420–421
 - Report Model Wizard, 417, 418
 - SSRS report builder role, 422–423
 - update Statistics page., 419
 - Set Logical Primary Key, 415
- Document mapping, 137–138
- Domain Controller, 374

E

- EmployeeServiceCost.rdl, 241
- Environmental Systems Research Institute, Inc. (ESRI), 4
- Excel 2002 and 2003, 11

F

- Filters, 170–171
- Format table, 154

G

- Gauge control, 179–181

H

- Health Insurance Portability and Accountability Act (HIPAA), 16
- Health-care application
 - Active Directory users and computers, 340, 341
 - analysis services data source, 368–369
 - browser, 341

Health-care application (*cont.*)
 Clinical Reports root folder, 345
 connection screen, 342, 343
 content filtering, 352
 content management permissions, 341
 content manager, 341
 Daily Activity report, 346
 data source authentication method, 345
 data warehouse database, 362
 db_datareader permissions, 345
 default security group, 348, 349
 error message, 347, 348
 graphical MDX query builder
 Average Referral to Admission report, 372
 Average Time to Admission chart, 371
 Design Mode toolbar button, 370
 dimensional and measure elements, 369, 370
 graphical design mode, 369
 MAX function, 370
 MIN function, 370
 output, 369, 370
 Referral to Admission Chart, 372
 warning message, 371
 Group Policy settings, 340
 HW_Analysis, 362
 Matrix-style reports, 366
 MultiDimensional expressions, 363
 My Reports feature, 341
 New Role Assignment form, 349
 Patient Referral cube, 362
 Average Referral to Admission report, 367
 branch, 364
 Browser tab, 364
 County, 364
 cube browser, 367
 Data Source Designer, 364
 Date of Admission, 364
 design mode, 364
 diagnosis, 364
 OLAP database, 364
 payer, 364
 Referral Source, 364
 Referral to Admission Chart report, 367
 structure, 362, 363
 Registered Nurse folder, 345
 Report Builder role, 341, 343, 344
 report objects, 346, 347
 reports publishing, 341
 RN_DS data source, 345, 346
 RNsecurity group, 340

RN Windows security group, 351
 roles and system roles, 342
 Rubik's cube, 364
 SSIS, 362
 test account, 351, 352
 Time to Admission, 365–366
 users task, 342
 windows integrated security option, 345
 HTML, 11
 text formatting, 5
 code output, 154
 Hyperlink
 actions, 143–147
 formatting and tooltips, 157

■ I, J, K

Integrated development environment (IDE), 13
 Error List window, 44
 Properties window, 44
 Report Data window, 44
 sample IDE configuration, 45
 toolbox, 44
 IntelliSense, 59
 Interactive sorting, 141–143
 ISNULL function, 34

■ L

Layer 2 tunneling protocol (L2TP), 330
 ListChildren method, 235, 236
 Lookup functions, 6

■ M

Matrix report creation
 Average Length of Stay, 437
 blank matrix report, 436
 data region, 435
 Discharge Reason field, 436
 Discharge Year parameter, 440–441
 filter data, 439–440
 Microsoft Excel, 435
 number formatting, 438
 Microsoft Directory Services, 54
 Microsoft Management Console (MMC), 39
 Microsoft Office Ribbon technology, 45
 Microsoft SharePoint 2010, 373
 Microsoft Word, 5, 11
 Microsoft's Reporting Services ad-hoc tool, 463

MIME HTML (MHTML), 11

Multivalued parameters

- Emp_Svc_Cost, 169
- fn_MVParam, string-parsing function, 169
- generated report, 170
- IN clause, 166, 167
- MVP, 166, 167
- NULL values, 166
- query/stored procedure, 165
- selection, 168
- table-valued function, 168
- WHERE clause, 165, 166, 169
- wrapping SQL statements, 168
- Year and Month parameters, 166

■ N

NULL rendering extension, 11

NULL values, 36

■ O

OverwriteDataSets, 221

OverwriteDataSources, 222

■ P

Parameterized stored procedure

- column and table aliasing, 35
- CREATE PROCEDURE command, 32
- database expansion, 32
- Emp_Svc_Cost, 32–34
- ISNULL function, 34
- performance and indexes, 34–35
- procedure testing, 35–36
- SELECT statement, 32

Patient census report

- admission and discharge information, 403
- BIDS 2008 R2, 405–406
- chart report (*see* Chart report creation)
- data source views (*see* Data source view creation)
- data source Wizard, 406–408
- Data Sources folder, 421–422
- health-care industry, 403
- matrix report (*see* Matrix report creation)
- process completed successfully, 419–420
- Report Builder 1.0 (*see* Report Builder 1.0)
- Report Builder 3.0 (*see* Report Builder 3.0)
- report model, 404, 420

table report (*see* Table report)

Patient Name detail textbox, 152

Patient Recertification Listing report, 298

Patient Survey Letter, 153, 156

Performance-tuning queries, 35

Placeholder Properties dialog, 155

Point-to-Point tunneling protocol (PPTP), 330

Portable Document Format (PDF), 11

■ Q

Query design, 19

advanced query

- COUNT and SUM functions, 28
- employee cost query, healthcare database, 27
- optimizing performance, 30–32
- ORDER BY PATID clause, 28
- required data output, 26
- SELECT clause, 27
- SQL Server machine, 28
- SSMS, 28–30

dataset configuration, 19

graphical query

- sample output, 26
- single-user execution scenario, 26
- SSMS, 23–25
- top ten patient diagnoses code, 25

parameterized stored procedure

- column and table aliasing, 35
- CREATE PROCEDURE command, 32
- database expansion, 32
- Emp_Svc_Cost, 32–34
- ISNULL function, 34
- performance and indexes, 34–35
- procedure testing, 35–36
- SELECT statement, 32

sample relational database

- Pro_SSRS database, 20
- schema design, 20–21
- sp_FieldInfo, 21–23

Query parameter, 158

■ R

Remote desktop protocol (RDP), 330

Report Builder 1.0, 46

BIDS/Visual Studio, 423

chart report (*see* Chart report creation)

- Report Builder 1.0, 4.6 (*cont.*)
 - ClickOnce application installation screen, 425
 - NET Framework 2.0, 424
 - Report Builder URL, 424
 - report designer, 423
 - Report Manager, 424
 - report model selection, 425–426
 - SQL Server 2008 R2, 424
 - table report template, 426–427
- Report Builder 2.0 Wizard, 45
 - connection string property, 450
 - data source and dataset, 454–457
 - default Ocean style, 452
 - Emp_Svc_Cost stored procedure, 450–451
 - Estimated_Cost field, 460
 - execution time and title, 462
 - finished report, 453
 - Insert tab, 457
 - layout selection screen, 452
 - Microsoft's Reporting Services ad-hoc tool, 463
 - New Table/Matrix wizard, 449
 - NULL values, 459
 - Preview button, 459
 - query design, 450
 - Ribbon technology, 447–448
 - Service Type Estimated Cost, 452
 - setting data source property, 450
 - SQL Server 2008 R2/SQL Server 2012, 449
 - SSRS 2008, 447
 - User ID, 461
 - Visit Count, 458
- Report Builder 3.0, 12, 14–15, 45, 219
 - BIDS/Visual Studio integrated tools, 218
 - data sources, 464
 - dataset caching, 464
 - drill, 465
 - Employee Service Cost by branch and patient state, 465–472
 - loaded report, 219, 220
 - Map report, 464
 - Pearl menu, 220
 - RDL file, 220
 - report items, 464
 - report parts, 464
 - save options, 221
- Report creation
 - adding a data source, 406–408
 - BIDS 2008 R2, 405–406
 - chart report (*see* Chart report creation)
 - data source view (*see* Data source view creation)
 - matrix report (*see* Matrix report creation)
 - patient census model, 404
 - Report Builder 1.0 (*see* Report Builder 1.0)
 - table report (*see* Table report)
 - user feedback, 402
- Report Definition Language (RDL), 1
 - Active Directory querying, 54
 - data source creation, 47–49
 - dataset creation, 49–53
 - expressions, 58–59
 - filters, 56–57
 - find-and-replace method, 45
 - OLE DB, 54
 - query and report parameters, 54–56
 - report adding, 46–47
 - updating and uploading
 - default parameter value, 288–290
 - Download and Replace links, 287
 - Notepad, RDL file, 287, 288
 - options menu, 285, 286
 - properties screen, 286, 287
 - ReportParameter, 287
 - XML-based schema, 45
- Report deployment
 - Actions link button, 393
 - application running, 239–242
 - Average Referral to Admissions, 392
 - configuration
 - Configuration Manager, 223–224
 - properties, 221–223
 - data source definition, 391, 392
 - PatRef_DS data source, 390
 - permission-related issues, 391
 - Report Builder 3.0 (*see* Report Builder 3.0)
 - Report Manager (*see* Report Manager)
 - report server web service (*see* Report server web service)
- Reports Library, 388
- rs.exe utility
 - CreateFolder method, 229
 - report and data source, 230
 - RSS file configuration, 226–227
 - RSS file contents, 227–229
 - VB .NET code, 226
- SharePoint Document Center main page, 388
- SharePoint document features, 390, 391
- SharePoint, rendered report, 392–394
- Solution Explorer, 224–225
- target URL properties, 389, 390

- Windows Credentials, 391
- Report layout, 61
 - list data region
 - design area, 63
 - free-form objects, 63
 - List Start, 63
 - patient group, 67, 68
 - Patient_Name grouping, 66, 67
 - preview, 64–66
 - RDL file, 68–69
 - Row Groups, 65
 - Tablix properties, 65, 66
 - ungrouped fields, 64
- Matrix data region, 86
 - cost estimation, 87
 - data field grouping, 85
 - Matrix Start.rdl, 85, 86
 - Matrix.rdl, 88
 - preview, 87
 - RDL output, 88
- page setup, 61–62
- rectangle data region
 - default currency format, 82
 - formatting report, 82
 - free-form container, 80
 - preview, 84
 - RDL output, 84
 - Rectangle Start.rdl, 81, 82
 - Rectangle.rdl, 85
 - Textbox report object, 80
- report objects, 62–63
- table data region
 - data organization, 75
 - detail row, 76
 - Estimated Cost expression, 78
 - Group Properties, 77, 78
 - preview, 79
 - RDL, 79
 - Row Group creation, 76, 77
 - Table Start.rdl, 75
 - Table.rdl, 80
- Textbox control
 - multiple format output, 74, 75
 - patient name value, 73, 74
 - plane text date formatting, 72, 73
 - Rich Text Formatting, 69
 - TextBox properties, placeholder, 70, 71
 - TextBox_Start report, 70
 - value properties, 71, 72
- Report management
 - data source, 290–292
 - execution auditing and performance analysis
 - built-in logging feature, 308
 - ExecutionLog Table, 309–311
 - performance monitoring, 313–315
 - Report Execution Log, 312–313
 - SSRS deployment, 308
 - SSRS logging, 309
 - RDL (*see* Report Definition Language)
 - report history, snapshots
 - caching, 296
 - data-driven subscriptions (*see* Data-driven subscriptions)
 - History tab, 295
 - report processing, 295–296
 - snapshot options, 293, 294
 - standard subscriptions (*see* Standard subscriptions)
 - types, 293
 - shared schedules
 - creation, 281–282
 - report configuring, 282–284
 - set up, 280–281
 - SSRS (*see* SQL Server Reporting Services)
- Report Manager, 12–13, 214
 - EmployeeServiceCost.rpt file, 215
 - manage option, 215, 216
 - properties modification, 216–218
 - RDL file, 215
 - Upload File option, 214, 215
- Report Page Layout (RPL), 11
- Report Processor, 10
- Report rendering, 11
 - buttonOK click Event Handler, 271
 - Class-Level Private Variables, 266
 - data source creation, 259
 - finalized Web report application, 277, 278
 - finished report, 273
 - GetItemParameters method
 - code, 267
 - Combo Item class, 270
 - ForRendering, 267
 - GetParameters_Load event method, 268–270
 - HistoryID, 267
 - Item, 267
 - Parameters dialog box, 271
 - ParameterValues, 267
 - ValidValue, 267–268
 - getParameters click Event, 272–273
 - GetParameters Form creation, 263–265
 - GetParameters_Load Event, 266

- Report rendering (*cont.*)
 - RenderReport Button Click method, 277
 - report designing steps, 259–262
 - report list, 276
 - report parameters form constructor, 266
 - Report Server web service, 244
 - calls, 244
 - categories, 263
 - ReportExecutionService, 262
 - ReportingService2010, 262
 - Report Viewer control, 243, 244
 - ASP.NET application, 245
 - button, 274
 - Default.aspx Partial Code Listing, 275
 - DropDownList, 274
 - Horizontal Rule, 274
 - local rendering, 245
 - reportViewerWeb, 274
 - ScriptManager, 274
 - server-side mode, 245
 - SOAP API, 243
 - URL access, 243, 245–246
 - credential parameters, 250
 - HTML Viewer commands, 247–248
 - HTML Viewer parameters, 247
 - optional path information, 246
 - readme.html file, 251
 - rendering access, 244
 - report document map, 251
 - report parameters, 246, 247
 - Report Server command parameters, 247–249
 - Report Viewer Web Part commands, 247, 250
 - rs:Format parameter, 251
 - syntax, 245
 - user-supplied parameters, 251
 - URL reporting parameters, 244
 - URL viewer application, 244
 - ViewerParameter method, 271
 - ViewerRVC.cs Form, 272
 - web service reference, 275, 276
 - Windows Forms SSRS 2008 viewer
 - application (*see* Windows Forms SSRS 2008 viewer application)
- Report library, 388, 389
- Report security
 - auditing, 356–358
 - components, 329
 - data encryption
 - complexity and reliability, 330
 - HTTPS, 330
 - IPSec, 330
 - Protected Health Information (PHI), 329
 - RDP, 330
 - VPN client systems, 330
 - WEP, 330
 - data sources
 - authentication, 340
 - connection options, 354–355
 - data storage securing, 337–339
 - filtering reports, 340
 - health-care application
 - Active Directory users and computers, 340, 341
 - browser, 341
 - Clinical Reports root folder, 345
 - connection screen, 342, 343
 - content filtering, 352
 - content management permissions, 341
 - content manager, 341
 - Daily Activity report, 346
 - data source authentication method, 345
 - db_datareader permissions, 345
 - default security group, 348, 349
 - error message, 347, 348
 - Group Policy settings, 340
 - My Reports feature, 341
 - New Role Assignment form, 349
 - Registered Nurse folder, 345
 - Report Builder role, 341, 343, 344
 - report objects, 346, 347
 - reports publishing, 341
 - RN Windows security group, 351
 - RN_DS data source, 345, 346
 - RNsecurity group, 340
 - roles and system roles, 342
 - test account, 351, 352
 - users task, 342
 - windows integrated security option, 345
 - HTTP traffic
 - capturing, 335–337
 - display filter rule, 332
 - display filter window, 332
 - network interface card, 331
 - network monitor, 330–332
 - PI data captured, 333
 - port 80, 331
 - report manager, 332
 - permissions settings, 340
 - SQL server permissions, 355–356
 - SSL certificate, 334–335

Pro SQL Server 2012 Reporting Services

Third Edition



Brian McDonald
Shawn McGehee
Rodney Landrum

Apress®

Pro SQL Server 2012 Reporting Services, Third Edition

Copyright © 2012 by Brian McDonald, Shawn McGehee, and Rodney Landrum

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-3810-2

ISBN-13 (electronic): 978-1-4302-3811-9

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

President and Publisher: Paul Manning

Lead Editor: Jonathan Gennick

Technical Reviewers: Rodney Landrum and Sherri McDonald

Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh

Coordinating Editors: Adam Heath, Stephen Moles and Kevin Shea

Copy Editor: Chandra Clarke

Compositor: Bytheway Publishing Services

Indexer: SPi Global

Artist: SPi Global

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at www.apress.com/bulk-sales.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

This book is dedicated to my beautiful wife and my bubbly best friend, Sherri, my two amazing children (Bailey and Kylie), my mother and father, and to everyone else who believed in me and supported me throughout the years.

–Brian K. McDonald

I dedicate this book to my wife, my children, and my parents.

–Rodney Landrum

I dedicate this book to my friends and family, who support me in everything I do.

–Shawn McGehee

Contents

■ About the Authors	xv
■ About the Technical Reviewers	xvi
■ Acknowledgments	xvii
■ Introduction	xix
■ Chapter 1: Introducing the Reporting Services Architecture	1
Understanding the Benefits of SSRS	2
SQL Server 2008 R2 and 2012 Reporting Services Enhancements	3
Report Builder/Data Modeler	3
SSRS 2012 Integration with Microsoft Office SharePoint.....	4
Tablix Report Properties	4
Enhanced Charting and Report Item Visualizations.....	4
Enhanced Performance and Memory Management	4
Embeddable SSRS Controls.....	5
HTML Text Formatting	5
Microsoft Word Rendering.....	5
Report Parts.....	5
Lookup Functions	6
Shared Datasets	6
SSRS and Business Intelligence	6
Business Intelligence Development Studio and SQL Server Data Tools	6
SQL Server Management Studio (SSMS)	6

SSRS Architecture	7
SSRS Databases	8
The SSRS Report Server	9
SSRS Web Service Interface.....	9
Authentication Layer.	10
The Report Processor	10
Data Processing.	10
Report Rendering.	11
Scheduling and Delivery.	11
Client Applications	12
Report Manager.	12
Business Intelligence Development Studio (BIDS) and SQL Server Data Tools (SSDT)	13
Command-Line Utilities	14
Report Builder.	14
Custom Clients.	15
Installing and Configuring.....	15
Deploying SSRS Securely	16
Summary	17
■ Chapter 2: Report Authoring: Designing Efficient Queries	19
Introducing the Sample Relational Database.....	19
Introducing the Schema Design	20
Knowing Your Data: A Quick Trick with a Small Procedure.	21
Introducing Query Design Basics.....	23
Creating a Simple Query Graphically.	23
Creating an Advanced Query	26
Testing Performance with SQL Server Management Studio (SSMS).	28
Optimizing Performance: Dividing the Load	30

Using a Parameterized Stored Procedure.....	32
Using ISNULL to Evaluate the Parameters.....	34
Query Performance and Indexes	34
Column and Table Aliasing	35
Testing the Procedure	35
Summary	36
■ Chapter 3: Introduction to Reporting Services Design with SQL Server Data Tools..	39
Exploring the Elements of BIDS	40
Setting Up a Basic IDE	43
Understanding Report Definition Language (RDL)	45
Adding a Report	46
Setting Up Data Sources and Datasets.....	47
Creating a Data Source.....	47
Creating a Dataset.....	49
Creating Other Data Sources	54
Configuring Parameters.....	54
Setting up Filters	56
Expressions	58
Summary	59
■ Chapter 4: Laying Out a Report.....	61
Setting Up Pagination	61
Using Report Objects	62
Implementing a List	63
Implementing a Textbox	69
Implementing a Table	75
Implementing a Rectangle.....	80

Implementing a Matrix.....	85
Summary	88
■ Chapter 5: Implementing Dashboard-Style Report Objects	89
Understanding the Chart Data Region	90
Implementing an Image.....	95
Implementing a Gauge.....	100
Implementing a Map.....	103
Implementing a Data Bar	110
Implementing a Sparkline.....	115
Implementing an Indicator.....	119
Summary	123
■ Chapter 6: Building Reports.....	125
Creating a Report with the Report Wizard	126
Building Reports from Scratch.....	129
Formatting the Output.....	133
Adding Subtotals.....	134
Adding Interactivity.....	136
Document Mapping	137
Visibility	138
Interactive Sorting	141
Hyperlink Actions.....	143
Adding a Bookmark Link.....	147
Adding a URL Link	149
Building the URL Link with a Report Parameter	150
Jumping to a Report.....	151
Adding Hyperlink Formatting and Tooltips	157

Setting Report Parameters with Stored Procedures.....	158
Working with Multivalued Parameters.....	165
Applying a Filter.....	170
Adding a Chart.....	171
Adding Tablix Elements.....	175
Configuring Report and Group Variables.....	177
Adding the Gauge Control.....	179
Adding the Final Touches.....	181
Summary.....	183
■ Chapter 7: Using Custom .NET Code with Reports	185
Using Embedded Code in Your Report.....	186
Using the ExceedMaxCosts Function.....	187
Using the ExceedMaxCost Function in a Report.....	190
Accessing .NET Assemblies from Embedded Code.....	194
Using Custom Assemblies with Your Report.....	195
Adding a Class Library Project to Your Reporting Solution.....	195
Deploying a Custom Assembly.....	198
Adding an Assembly Reference to a Report.....	202
Debugging Custom Assemblies.....	207
Troubleshooting Your Project.....	210
Summary.....	211
■ Chapter 8: Deploying Reports	213
Using Report Manager.....	214
Using Report Builder 3.0.....	218
Using BIDS and Visual Studio 2012.....	221
Configuring Report Deployment Options.....	221

Setting Up Deployments Using the Configuration Manager.....	223
Deploying Reports Through the Solution Explorer.....	224
Using the rs.exe Utility.....	226
Using the Report Server Web Service.....	230
Accessing the Web Service	231
Laying Out the Form	233
Coding the Form	234
Allowing Users to Enter a Server Name.....	235
Populating the TreeView Control With a List of Folders.....	235
Opening the RDL File and Uploading It to the Server.....	237
Running the Application.....	239
Summary	242
■ Chapter 9: Rendering Reports from .NET Applications	243
Implementing URL Access	245
URL Report Access Path Format.....	246
URL Parameters and Prefixes.....	246
Report Parameters	247
HTML Viewer Commands	247
Report Server Command Parameters.....	248
Credential Parameters.....	250
Report Viewer Web Part Commands.....	250
Example URLs.....	251
Integrating SSRS 2012 with .NET Applications.....	251
Building a custom Report Viewer Using the WebBrowser Control	251
Building the Report Viewer Using a Report Viewer Control	255
Rendering the Report Locally	259
Creating the Report's Data Source	259

Designing the Report.....	260
Using the Report Server Web Service.....	262
Web Services Method Categories.....	263
Creating the GetParameters Form	263
Coding the Report Parameters Form	266
The GetParameters_Load Event	266
Calling the Web Services GetItemParameters Method	267
Rendering the Final Report.....	271
Building the Report Viewer in ASP.NET	274
Summary	278
■ Chapter 10: Managing Reports	279
Exploring Management Roles in SSRS Deployment	279
Managing Content.....	280
Setting Up Shared Schedules	280
Creating a Shared Schedule	281
Configuring a Report to Use a Shared Schedule	282
Updating and Uploading the RDL File Using Report Manager	284
Setting Up a Data Source for the Report.....	290
Altering Report Data Sources	292
Creating Snapshots for the Report History	293
Processing Reports and Performing Caching	295
Managing Subscriptions	297
Managing Standard Subscriptions	298
Creating a Standard Subscription.....	299
Configuring the Subscription	301
Managing Data-Driven Subscriptions.....	303
Designing the Subscription Query	303
Creating the Data-Driven Subscription	305

Performing Execution Auditing and Performance Analysis	308
Configuring SSRS Logging.....	309
Transforming the ExecutionLog Table.....	309
Designing the Log Report	312
Monitoring Performance	313
Controlling SSRS Programmatically.....	315
Controlling SSRS with SOAP	315
Controlling SSRS with WMI.....	326
Summary	327
■ Chapter 11: Securing Reports.....	329
Encrypting Data	329
Introducing Encryption.....	330
Securing Network Traffic Using SSL.....	330
Analyzing HTTP Traffic.....	330
Applying an SSL Certificate	334
Capturing HTTPS Traffic	335
Securing Data Storage in SSRS.....	337
Setting Up Authentication and User Access to Data.....	339
Introducing SSRS Roles	340
Testing SSRS Role Assignments	344
Filtering Report Content with User!UserID.....	352
Setting Data Source Security.....	354
Setting SQL Server Permissions	355
Auditing Reports	356
SSRS Auditing.....	356
Introducing Log File Auditing.....	358
Summary	359

■ Chapter 12: Delivering Business Intelligence with SSRS.....	361
Building SSRS Reports for SQL Analysis Services	362
Using an Analysis Service Cube with SSRS.....	367
Setting Up the Analysis Services Data Source	368
Working with the Graphical MDX Query Builder	369
Incorporating SSRS with Microsoft SharePoint 2010	373
Installing SharePoint 2010 and SQL Server 2012 on a Stand-Alone Server	373
Installing SharePoint 2010.....	374
Installing SQL Server 2012 Reporting Services in SharePoint Mode.....	375
Configuring SharePoint 2010.....	377
Installing and Starting the Reporting Services SharePoint Service.....	379
Creating a New Reporting Services Service Application	381
Configuring Reporting Services Integration with SharePoint	384
Deploying Reports in a SharePoint-Integrated SSRS Installation	388
Creating a Simple Dashboard to Display SSRS Reports	394
Creating Data Alerts.....	397
Summary	399
■ Chapter 13: Creating Reports Using Report Builder 1.0, 2.0, and 3.0	401
Getting User Feedback	402
Introducing the Report Model	404
Create a Report Model using BIDS 2008 R2	405
Adding a Data Source	406
Creating a Data Source View	409
Creating a Report Model.....	417
Creating Reports with Report Builder 1.0	423
Creating a Table Report	427
Creating a Matrix Report.....	435

Creating a Chart Report	441
Creating Reports with the Report Builder 2.0 Wizard	447
Creating Reports with Report Builder 2.0	454
Creating Reports with Report Builder 3.0	464
Report Parts	473
Summary	482
■ Index	483

About the Authors



■ **Brian K. McDonald**, MCDBA, MCSD, is a business intelligence and data warehouse geek working for Acosta, the top sales and marketing company in the US. Brian was born and raised in Cincinnati, Ohio but he currently lives in Jacksonville, Florida with his beautiful wife and his two little ninjas, Bailey and Kylie. As the former owner of Business Enterprise Solution Technologies and an independent consultant, Brian has experiences in architecting data warehouses, data modeling, database administration, SQL Server Reporting Services, Integration Services, Analysis Services, and PowerPivot. He has also worked as an application developer, network administrator, and database administrator throughout his 13 years in the IT industry. A characteristic groomed while serving his country in the United States Marine Corps is the pride Brian takes in all his projects and he diligently works to not only meet expectations, but greatly exceed them. He continually strives to improve his skill-set, both professionally and personally. In addition to having a Business

Management and Computer Information Systems, he is a Microsoft Certified Solution developer (MCSD) and a Microsoft Certified Database Administrator (MCDBA). Brian is an active member in the SQL Server community. He enjoys giving back to the SQL Server community through training, virtual and public speaking engagements, as well as by writing articles and blogs for many industry leading websites like SQLBIGeek, SQLServerCentral, and SQLServerPedia. You may have seen Brian presenting at the inaugural SQL Rally in Orlando, FL, or at one of the many SQL Saturdays, Code Camps, or User Groups that he has been a part of throughout the years. In those moments when he is not giving back to the community or working, Brian enjoys "geeking" out with his wife Sherri and allowing his children to practice their UFC style moves on him.



■ **Shawn McGehee** is a DBA and manager for a large insurance company in the US. He has been working with SQL Server in some fashion since 7.0 and has been working in the IT field since 1998. Starting as a developer, he has gradually made the move to a full time administration role, but still enjoys the challenges he can solve using his development background whenever he can. He is active in the SQL Server community and enjoys speaking at user groups, SQL Saturdays and the odd Code Camp. He is currently running the Orlando SQL Server User Group OPASS and has been involved with the SQL Server community for the last few years. You can usually find him roaming about at a SQL Saturday in the SouthEast, wherever it may be.

About the Technical Reviewers



■ **Rodney Landrum** has worked with SQL Server longer than he can remember. He writes regularly about technologies, including Integration Services, Analysis Services, and Reporting Services. (Rodney is the original author of this book). He has authored *SQL Server Tacklebox* and three Reporting Services books. He contributes regularly to SQLServerCentral, SQL Server Magazine, and Simple-Talk. His day job involves overseeing a large SQL Server infrastructure in Orlando. He swears that he owns the phrase "Working with Databases on a Day to Day Basis." Anyone who disagrees is itching to lose an arm wrestling match.



■ **Sherri McDonald** is a BI Developer who has specialized in SQL Server Reporting Services as a trainer of Reporting Services and the fundamentals of TSQL. She also is known for her Reporting Services DVD, co-produced with SSWUG. She has served many roles, with more than 14 years in the service industry, while most recently focusing on the Microsoft BI Stack. Sherri is an active blogger, a member of her local SQL Server User's Group, and has presented at SQL Saturday and Code Camp events throughout Florida. Sherri was born and raised in Cincinnati, Ohio and currently lives in Jacksonville, Florida with her husband Brian and two children. When not learning the latest in technology, Sherri enjoys movies, going on cruises, and other travels with her friends and family.

Acknowledgments

First and foremost, I must thank my bubbly best friend, fellow SQLBIGeek, biggest fan, and the mother of my brilliant children—my beautiful wife Sherri. Without her, I would never have achieved many of the successes that I have achieved. Sherri, you are amazing and I could not have done this without your support, guidance, patience, and endless love. With that being said, I would also need to thank my remarkable children, Bailey and Kylie. Thank you for all of the understanding, support, and patience while I locked myself away in the dungeon for so many hours writing this book. Without you two being such wonderful children, I could not have completed this book.

Next, I would like to thank my friend, coauthor, and technical reviewer, Rodney Landrum, for all of his support and belief in me throughout the years. I am so thankful to be able to call you a friend and I look forward to any future opportunities that we have to mash our brains together. Thank you!

To another friend and coauthor, Shawn McGehee: without all of your hard work and dedication to putting out another great release, this book would not have come to a close. Thank you so much for all of your ideas and contributions.

I would like to thank the Apress editorial team, Kevin Shea, Stephen Moles, Adam Heath, and an extra special thanks to Jonathan Gennick, for all of their encouragement and guidance throughout this project. Through the hundreds of email messages and conversations, you have expertly moved this book along to completion and I look forward to getting to meet you all in person. It has truly been a pleasure working with each of you. Thank you!

To my brilliant and humble SQL Server DBA friend and cousin, Chad Tucker: without that kick in the butt many years ago, I would not be in Jacksonville, Florida. Without that nudge, I wouldn't have pursued public speaking, blogging, and eventually even authoring my first publishable work of art (this book). From the bottom of my heart to the tip of my tongue, thank you!

Another person that I must provide a special thanks to is my good friend Marc Munago. He probably doesn't know this, but he has been a great inspiration to me the last few years with his leadership traits and his ability to turn a penny into a dollar. He has been a true friend and mentor to me. We often have great "idea" sessions where we bounce goals, plans, and aspirations back and forth. Your support is very much appreciated and I thank you!

To my awesome neighbors, Eric and Cathy Eng! Eric is an intelligent entrepreneur, Engineer, and an amazing father. Cathy, being a long time writer, has helped me out several times; she owns and runs a company called Resume Rocketeer (www.resumerocketeer.com). Sherri and I always enjoy hanging out with you. Thank you both for all of the late night talks and friendship.

To Scott Gleason, a great friend, supporter, and a huge fan! I am thankful for all of your encouragement and your verbal marketing throughout this process, and the last several years for that matter! You are a great friend and I thank you from the bottom of my heart!

Let me not forget to thank Cricket Weaver and Patrick Barr, who allowed me to fumble through many proof of concepts and thorough testing of Reporting Services, while being a DBA consultant many moons ago! Thank you for the patience and leadership that you both showed when we ramped up on the product. Without that leadership, I probably would have thrown in the towel.

To all of the amazing employees at Acosta, especially the Data Warehouse and Business Intelligence (BI) team with whom I work. Not only are you are the most brilliant team of BI professionals that I have worked with over the years, you are also the most fun to work with. I truly enjoy coming into the office each and every day, as I look forward to what each day might bring. You are the *crème de la crème* of the Information Technology world. Thank you!

A very special thank you goes out to Tom and Mary Beth Ottke, who have been two of the most influential people in my life. You both are such intelligent, kind, loving, and wonderful people. You accepted me into your family many years ago and you helped me become the man that I am today. Without you, I probably wouldn't have even taken the first step in this journey. It is you who planted the seed that gave me belief—not only in myself, but in Him as well! Thank you for all that you have done for me.

To you, the reader of this book, I hope that we (the authors) will teach you about the latest edition of SQL Server Reporting Services. We thank you for spending your hard-earned money on our book and hope that you find it a valuable resource to have sitting at your fingertips.

Last, and certainly not least, I would like to thank my mother and father for everything you have done for me throughout the years. Your love, guidance, and words of encouragement have always been an inspiration to me. Even when you thought that I wasn't listening, your advice and wisdom were always well received. Without all of the leading by example and your hard work, I would not have followed through with this book.

—Brian K. McDonald

I would like to thank everyone who has helped make this book a reality. Rodney Landrum and Brian McDonald are both great collaborators and I couldn't ask for a better pair of co-authors. The staff at Apress has been excellent as well. With their support and guidance, we have been able to produce a work that we hope will help guide readers into the realm of SSRS. I would also like to thank all of the friends and family who have been supportive of the process along this long road.

—Shawn McGehee

I would like to thank Brian McDonald and Shawn McGehee for their contributions to this edition of the book and for tolerating me over the years. Through them, I have met many others who share our passions for SQL Server and community. This book is but one outlet of their service and dedication to these technologies and to the people like us who work and play with data every day. Well, except for holidays and most weekends. Cheers to them!

I would also like to acknowledge the efforts of the Apress staff, who patiently worked with us to mold this edition into a book we can be proud of.

—Rodney Landrum

- SSRS roles, 339
- system administrator, 350
- system user, 350
- user authentication, 339
- windows integrated authentication, 340
- Report Server, 8, 9
- ReportServerTempDB, 8
- Report server web service, 315–316
 - access, 231–233
 - calls, 244
 - categories, 263
 - CreateCatalogItem method, 230
 - CreateReportEditSession method, 231
 - form coding, 234
 - form layout, 233–234
 - RDL file, 237–239
 - ReportExecutionService, 262
 - ReportingService2010, 262
 - server name, 235
 - SOAP API, 230
 - TreeView Control population, 235–237
 - Windows Forms application, 230
- Report Wizard
 - Corporate style, 128
 - data source, 126
 - generated report, 129
 - group and details selections, 128
 - Query String area, 127
 - tabular/matrix form, 127

■ S

- Secure sockets layer (SSL), 330
- ServiceTypeID, 32
- SQL Server Data Tools (SSDT), 6, 13–14
- SQL Server Integration Services (SSIS), 1, 309, 362
- SQL Server machine, 32
- SQL Server Management Studio (SSMS), 6–7
 - advanced query, testing performance, 28–30
 - graphical query, 23–25
- SQL Server Profiler, 22
- SQL Server Reporting Services (SSRS), 90, 91
 - advantages
 - BI strategy, 3
 - cost, 2
 - customizable, 2
 - SharePoint portal server, 3
 - standard platform, 2
 - subscriptions, 2
 - Web-enabled solution, 2
- architecture
 - authentication layer, 10
 - components, 7, 8
 - data processing, 10–11
 - databases, 8–9
 - Report Processor, 10
 - report rendering, 11
 - Report Server, 9
 - scheduling and delivery, 11–12
 - Web service interface, 9–10
- Business intelligence (*see* Business intelligence)
- charts and visualizations, 4
- client applications
 - BIDS/SSDT, 13–14
 - command-line utilities, 14
 - custom clients, 15
 - installation and configuration, 15–16
 - Report Builder 3.0, 14–15
 - Report Manager, 12–13
 - secure deployment, 16–17
 - tools, 12
- controlling with SOAP
 - Class-Level Private Variables, 317
 - ComboItem class, 319
 - CreateSubscription method, 319
 - ListSchedules method, 318
 - Namespaces importing, 317
 - PickSchedule Constructor, 318
 - PickSchedule.cs class file, 316, 317
 - PickSchedule_Load event, 317
 - PickSchedule_Load method, 319
 - report delivery, 324–326
 - report scheduler, 320–323
 - Report Server Web service, 315, 316
 - subscription functionality, 316
- controlling with WMI, 326
- data modeler, 3
- deployment, 279, 308
- embeddable controls, 5
- HTML text formatting, 5
- logging, 309
- lookup functions, 6
- memory management, 4
- Microsoft Word rendering, 5
- performance enhancement, 4
- Report Builder, 3, 422–423
- report parts, 5
- roles, 339
- shared datasets, 6

- SQL Server Reporting Services (*cont.*)
 - SSRS 2012 Integration with Microsoft Office
 - SharePoint, 4
 - standardization, 1
 - Tablix report properties, 4
- Stand-alone installation, 373
 - final configuration process, 384–388
 - installation steps, 374
 - New Reporting Services Service Application, 381–384
 - Reporting Services SharePoint Service, 379–381
 - SharePoint 2010
 - configuring, 377–379
 - installation, 374–375
 - SQL Server 2012, 375–377
- Standard subscriptions
 - configuring, 301
 - creation, 299–300
 - Patient Recertification Listing report, 298
 - Pro_SSRS data source, 298
- Subtotals, 134–136

■ T

- Table report creation
 - Branch Name field, 432–433
 - Branch Name group, 433–434
 - data region, 427
 - Define Formula dialog box, 429–430
 - dragging and dropping fields, 432
 - Page Setup dialog box, 428–429
 - Pat Last Name field, 430
 - patient face sheet, 427–428, 430–431
 - Pro_SSRS folder, 434
 - ReportServer database, 434–435

- sorting and grouping options, 431–432
 - user feedback, 427
- Table-valued function, 168
- Tablix elements, 175–177
- Tagged Image File Format (TIFF), 11
- TargetDataSetFolder, 222
- TargetDataSourceFolder, 222
- TargetReportFolder, 222
- TargetReportPartFolder, 222
- TargetServerURL, 222
- TargetServerVersion, 222
- Trx.ServicesTblID field, 178

■ U

- URL link, 149–151

■ V

- Virtual Private Network (VPN), 17, 330
- Visual Basic .NET (VB.NET), 1

■ W, X, Y, Z

- Windows Forms SSRS 2008 viewer application
 - Report Viewer control
 - advantage, 255
 - coding, 257–258
 - properties and methods, 255
 - ViewerRVC.cs form, 256–257
 - WebBrowser control
 - coding, 253–255
 - ViewerWBC.cs form creation, 252–253
- Windows management interface (WMI), 326
- Wireless encryption protocol (WEP), 330