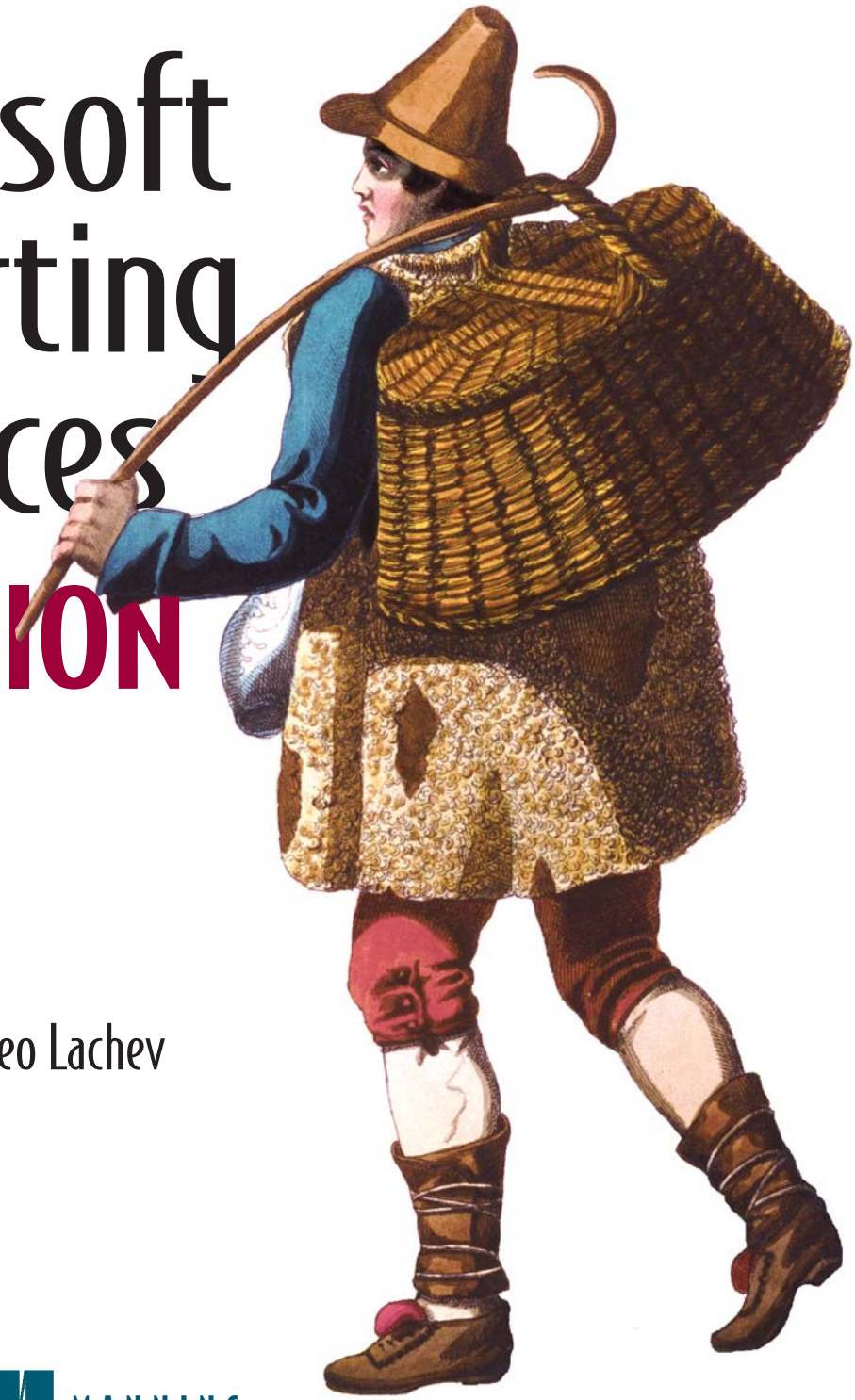
SAMPLE CHAPTER

Microsoft Reporting Services IN ACTION



Teo Lachev

MANNING

Microsoft Reporting Services in Action

by Teo Lachev

Chapter 1

brief contents

CHAPTER 1

Introducing Microsoft Reporting Services

So much information, so little time ... the character "Poison Ivy" would likely say if the Batman saga was taking place in today's enterprise.

We all know that the dot.com boom is history and so are the lavish IT budgets. In the doldrums of the economic recovery, organizations tend to spend their money on streamlining internal processes to gain a competitive advantage. According to Microsoft, today's information workers spend as much as 80 percent of their time gathering information, with only 20 percent left to analyze it and make a decision. In many organizations, such requests consume significant IT and development resources. Too often, Excel spreadsheets are the prevalent reporting tools today and manual data entry or "pencil-pushing" is among the top reasons for inaccurate data and wrong decisions. Aware of these issues, Microsoft initiated the Microsoft SQL Server 2000 Reporting Services project at the beginning of the new millennium, with a bold vision to "enable employees at all levels of an organization to realize the promise of Business Intelligence to promote better decision making."

This chapter provides a panoramic view of Reporting Services (RS). Throughout the rest of this book I will use the terms *Reporting Services* and *RS* interchangeably. You will see

- Why RS is such a compelling choice for enterprise reporting
- The main parts of the RS architecture
- The report-generation process and report lifecycle
- The steps for creating your first RS report

1.1 WHAT IS RS?

Regardless of the alphabet soup of terms and acronyms that are popping up like daisies almost every day and that have probably become a part of your IT vocabulary—terms such as BI (business intelligence), OLAP (online analytical processing), data mining, DSSs (decision support systems), EISs (executive information systems), digital dashboards, enterprise portals, and enterprise data buses—the purpose of enterprise reporting is to simply "get out" what was "put in." Therefore, for many applications, reporting represents the last, and often most important, stage of the IT pipeline.

To clarify the last point, let's consider a typical scenario that RS can address effectively. Let's say that an organization has built a web portal for submitting orders online. As the business grows, the same organization may need to implement a reporting infrastructure to analyze sales data and understand its business, for example, to find out the top-selling products, customer demographics, and so forth. To accomplish this goal, the organization could leverage RS.

We use the term *report* to refer to the web-based or saved-to-file counterpart of a standard paper-oriented report. For example, an organization may want to give its customers an option to generate various reports online—an Order History report, for instance. Web reporting has traditionally been difficult to implement. Even more difficult has been exporting reports to different file formats. RS solve both problems elegantly, for two reasons. First, out-of-the-box RS is web-enabled. Second, most popular export formats are natively supported.

1.1.1 Why do we need RS?

Ironically, despite the important role that reporting plays in today's enterprise, creating and distributing reports have been traditionally painstaking and laborious chores. To understand why we need RS, let's analyze the reporting problem space.

Table 1.1 lists some of the most pressing issues surrounding the reporting arena and how RS addresses them.

Table 1.1 How Microsoft RS deals with the reporting problem space

Reporting Need	How RS addresses it?
Report authoring can be labor intensive.	By using the powerful Report Designer, you can author reports as easily as you can with Microsoft Access.
Centralized report management is needed.	RS enables you to save your reports in a single report repository.

continued on next page

Table 1.1 How Microsoft RS deals with the reporting problem space *(continued)*

Reporting Need	How RS addresses it?
Reports need to be distributed to various destinations.	RS supports both on-demand and subscription-based reporting. Reports can be requested on-demand by Win-Form and web-based applications. Alternatively, reports can be distributed to a list of subscribers.
Reports often need to be exported in different electronic formats.	RS supports many popular export formats out of the box.
Proprietary nature of reporting tools doesn't allow you to extend them.	RS has a flexible architecture that allows you to extend RS capabilities by writing custom code.
Reports need to be secured.	RS offers a comprehensive security model that administrators can leverage to enforce secured access to reports by assigning users to roles. When the default Windows-based authentication is not a good fit, it can be replaced with custom security implementations.
Enterprise reporting solutions can be costly.	To minimize cost, RS is bundled and licensed with SQL Server. If you have a licensed copy of SQL Server 2000, you may run RS on the same server for no additional license fee.

Depending on your particular situation you may find other compelling reasons to target RS as your reporting platform of choice. We revisit the RS features throughout this chapter.

Supported report types

Your reporting requirements may call for authoring various types of reports that differ in complexity. For example, your users may request that a large report include a document map for easy navigation. RS lets you design a variety of report types, as listed in table 1.2.

Table 1.2 RS supports various report types

Report Type	Purpose	Example
Tabular	Displays data in a table format with a fixed number or rows and columns.	Excel-type reports
Freeform	Data regions are positioned arbitrarily on the page by the report author.	Invoice-invoice details report
Chart	Presents data graphically.	Employee performance chart
Crosstab (matrix)	Data is rotated to present row data as columns.	A report that shows products on rows and time on columns
Drilldown	Includes expandable sections.	A company performance crosstab report where product can be expanded by category and brand

continued on next page

Table 1.2 RS supports various report types *(continued)*

Report Type	Purpose	Example
Drillthrough	Generated from clicking on a hyperlink.	Customer Order History with hyperlinks on the order identifier to show the order details report
Interactive	Includes interactive features, such as document maps, hyperlinks, visible-on-demand sections, and so forth.	Adobe Acrobat–type reports with document maps on the left side

Although most popular reporting tools support many of the report types shown in table 1.2, RS makes the report-authoring process as easy as working with Microsoft Access reporting functionality. For example, report authors can drag and drop items to define the report's appearance.

Now that we understand what RS is, let's see how it fits in the Microsoft BI vision.

1.1.2 How is RS implemented?

Microsoft released version 1.0 of RS at the beginning of 2004 as an add-on to Microsoft SQL Server 2000. At a very high level, RS can be defined as a server-based platform for authoring, managing, and distributing reports. We discuss the RS architecture in more detail in a moment. For now, note that RS is integrated with and requires several other Microsoft products, including:

- Windows 2000 or above as a server operating system
- Microsoft SQL Server 2000 (with Service Pack 3a) and above
- Internet Information Server (IIS) 5.0 or above
- .NET Framework 1.1
- Visual Studio .NET 2003 for report authoring and testing

For more information about installing RS, please refer to appendix A.

RS editions

To address different user needs, RS is available in several editions, as you can see by looking at table 1.3.

Table 1.3 RS supports editions to meet various reporting needs

Edition	Choose when…
Standard	You need to install RS on a single computer. The Standard edition doesn't support clustered deployment to load-balance multiple RS instances.
Enterprise	You need all RS features, including load balancing.
Developer	You have to integrate RS with client applications or extend its capabilities by writing .NET code. The Developer edition supports the same feature set as the Enterprise edition, but it is for use as a test and development system, not as a production server.
Evaluation	You need to evaluate RS. The Evaluation edition expires after 120 days.

For more information about how the RS editions differ, refer to the product documentation or the "Reporting Services Features Comparison" section in the RS official website at http://microsoft.com/sql/reporting/productinfo/features.asp.

For information about RS licensing requirements, visit the "How to License Reporting Services" page at http://www.microsoft.com/sql/reporting/howtobuy/howtolicensers.asp.

1.1.3 RS and the Microsoft BI platform

RS is positioned as an integral part of Microsoft's business intelligence (BI) platform. This platform is a multiproduct offering whose goal is to address the most common data management and analysis challenges that many organizations face every day, such as analyzing vast volumes of data, trend discovery, data management, and of course, comprehensive reporting.

During the RS official launch presentation on January 27, 2004, Paul Flessner, Microsoft senior vice president of Enterprise Services, outlined the place of RS in the Microsoft BI platform offering, as shown in figure 1.1.

Table 1.4 outlines the purpose of the major building blocks within the Microsoft BI platform.

Most of you have probably used more than one of these products in the past to solve your data management and analysis needs. Indeed, most of them have been around for a while. What was missing was a product for authoring, managing, and

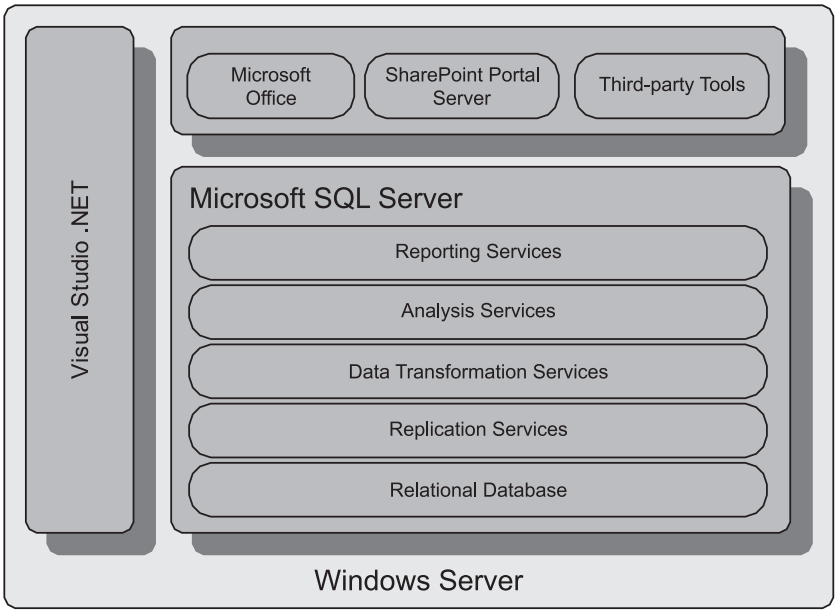


Figure 1.1 The Microsoft BI platform consists of several products layered on top of the SQL Server database engine and addresses various data management and reporting needs.

Table 1.4 The key Microsoft BI platform components

Component	Purpose
Microsoft SQL Server	A relational database to store data
Analysis Services	An analytical processing (OLAP) engine
Data Transformation Services	Tools for extracting, transforming and loading data
Reporting Services	Server-based reporting platform for report authoring, management and delivery
Replication Services	Replicates data to heterogeneous data sources
Microsoft Office	Desktop applications for data analysis and reporting
SharePoint Portal Server	Business Intelligence collaboration
Visual Studio.NET	A development tool to create .NET-based applications, including analytical and reporting solutions.

generating reports that could be easily integrated with all types of applications. RS fills the bill nicely.

Having introduced you to RS, let's take a panoramic view of its features to understand why it can be such a compelling choice for enterprise reporting.

1.2 RS AT A GLANCE

Even in its first release, RS offers a broad array of features that can address various reporting needs:

- *Information workers can leverage RS to author both standard ("canned") reports and reports with interactive features.* Here, we use the term "standard" to refer to reports that display static data. An interesting aspect of RS is that your reports can include a variety of features that provide interactivity to users. For example, the end user can show or hide items in a report and click links that launch other reports or web pages.
- *Third-party vendors can target RS to package reports as a part of their applications.* For example, if customers have RS installed, the vendor setup program can upload the report files to the Report Server. You'll see this done in chapter 2. Note that the next version of RS is expected to include stand-alone controls for generating reports directly from report files and will not require RS to be installed.
- *Organizations can use RS to report-enable their business-to-business (B2B) or business-to-consumer (B2C) applications.* For example, an organization can selectively expose some of its data in the form of reports to its business partners. You'll see an example of a similar integration scenario in chapter 11.

Let's now get a glimpse of the RS landscape and observe some of RS's most prominent landmarks. Don't worry if you find you are not getting the Big Picture yet. In section 1.3, we take a closer look at the main pieces of the RS architecture.

1.2.1 Authoring features

As a report author, with RS you have several choices for creating reports. We discuss each of these options in detail in chapter 2. For now, we'd like to introduce you to the Report Designer; this will likely be the option that you will use most of the time for report authoring.

Introducing the Report Designer

Using the Report Designer graphical environment, you can create reports of different types, such as crosstab drilldown reports, like the one shown in figure 1.2.

RS doesn't restrict your report-authoring options to static paper-oriented reports. Instead, you can make your reports more versatile and easy to use by adding interactive features, such as expandable sections, hyperlinks, and document maps. Given its tight integration with the Visual Studio. NET integrated development environment (IDE), the Report Designer provides you with access to all report design features as well as team development features, such as source code management.

About the Report Definition Language

At this point, you may be wondering what an RS-based report file looks like and how it is stored. RS saves the report as an Extensible Markup Language (XML) file that is described in a Report Definition Language schema.

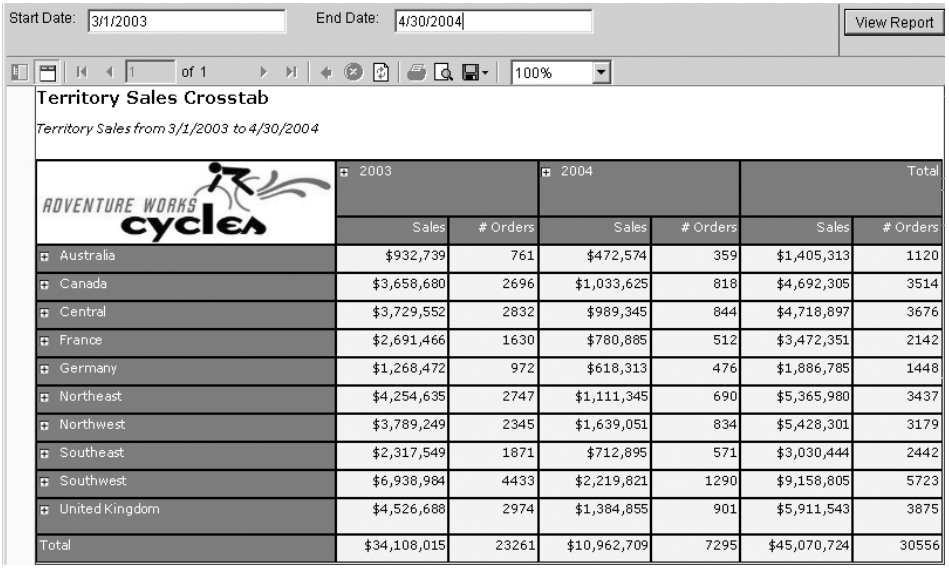


Figure 1.2 With RS you can create various types of reports, including drilldown crosstab reports like this one.

DEFINITION A *report definition* contains report data retrieval and layout information. The report definition is described in an XML schema, called the Report Definition Language (RDL).

Saving reports as XML-based report definition files offers two main advantages:

- *It makes the report format open and extensible.* Using the XML-based RDL format is beneficial for achieving interoperability among applications and vendors. Microsoft is working with other industry leaders to promote RDL as an XML-based standard for report definitions. Visit the RS official website (check the Resources section for the link) for a list of Microsoft RS partners.
- *It makes the report portable.* For example, you can easily save the report to a file and upload it to another Report Server. In chapter 2 you'll see how a third-party reporting tool leverages this feature for ad-hoc reporting.

If you use the Report Designer to create your report, its definition will be automatically generated for you. However, just as you don't have to use Visual Studio .NET to write .NET applications, you can write the report definition using an editor of your choice, such as Notepad, or generate it programmatically (as you will see in chapter 2). Of course, the Report Designer makes authoring reports a whole lot easier. Third-party tools will most likely emerge at some point to provide alternative RDL editors.

1.2.2 Management features

RS facilitates report management by storing reports and their related items in a central report catalog. To deploy and manage a report, you need to upload it to the report catalog. When this happens, it becomes a *managed* report.

DEFINITIONS Throughout the rest of this book we will use the terms *report catalog* and *report repository* interchangeably to refer to the RS Configuration Database. For more information about this database, refer to section 1.3.2.

A *managed report* is a report that is uploaded to the report catalog.

For .NET developers, the term "managed" has nothing to do with .NET managed code, although the pattern is the same. While .NET managed code runs under the supervision of the .NET Common Language Runtime (CLR), a managed report is generated under the control of the Report Server.

You may wonder what really happens when a report is uploaded to the report catalog. At publishing time, the Report Server parses the report definition (RDL), generates a .NET assembly, and stores the assembly in the Report Configuration Database for the report. The RDL file is never used again. When the report is processed, the assembly is loaded and executed by the Report Server.

A report can include other items, such as images and data source–related information. These report-related items are also stored in the report catalog. Finally, the report catalog captures additional information, called *metadata*, associated with reports. For

example, just as you can organize physical files in folders, RS allows you to organize reports in folders.

DEFINITION The report *metadata* describes additional configuration information associated with a report, such as security permissions, the parent folder, and so forth.

RS offers centralized report management that administrators will appreciate. To simplify the administration of the report catalog, RS comes with a tool called the Report Manager. The Report Manager is implemented as a web-based application, and as such it is easily accessible. This tool empowers you to manage just about any aspect of the report repository, including

- Report information and metadata, such as the folder structure and report properties
- Data sources from which the report will draw data
- Report parameters (for parameterized reports)
- Security

1.2.3 Delivery features

Reports hosted under RS can be delivered using on-demand ("pulled") delivery or subscribed ("pushed") delivery. The more common scenario is on-demand delivery, where the user requests the report explicitly. As a report author, you don't have to do anything special to web-enable your report because RS does this for you once it is uploaded to the report catalog.

The "pushed" delivery option alone can justify implementing RS. This option gives end users the ability to subscribe to reports, so reports will be sent to them when a certain event is triggered—when a timing event triggers, for instance, for report subscriptions based on a schedule. As another example, a financial institution could allow its customers to opt in and subscribe to certain reports of interest, such as a monthly bank statement. Then, at the end of the month, the bank statement report could be generated and sent to users via e-mail.

We'll discuss the report-delivery process in more detail in section 1.5.

1.2.4 Extensibility features

An important characteristic of every enterprise-oriented product, such as RS, is that it has to be easily extendable. Simply put, extensibility relates to the system's ability to accommodate new features that are built out of old ones. One of the things I like most about RS is the extensibility features it includes by virtue of its open and flexible architecture. Developers can easily extend RS by writing .NET code in their preferred .NET language. Specifically, you can extend RS in the following areas:

- *Custom .NET code*—.NET developers can enhance reports programmatically by writing .NET custom code. Chapter 6 demonstrates how you can add forecasting features to your reports by using prepackaged code in the form of .NET assemblies.

- *Data processing extensions*—Out of the box, RS can connect to any data source that has an ODBC or OLE DB provider. In addition, you can write your own custom data extensions to report off other data structures, as chapter 15 illustrates.
- *Delivery extensions*—Out of the box, subscribed reports can be delivered via e-mail or file share delivery extensions. Developers can write their own delivery extensions to deliver the report to other destinations, such as to web services, as you'll learn in chapter 15.
- *Security extensions*—By default, RS uses the Windows-based security model to enforce restricted access to the report catalog. If Windows-based security is not an option, you can replace it with custom security models. You'll see an example of how this could be done in chapter 15, where we'll implement custom authentication and authorization for Internet-oriented reporting.
- *Rendering extensions*—Generating reports in other export formats than the ones supported natively can be accomplished by writing custom rendering extensions. See section 1.4.2 for more information about the supported export formats.

1.2.5 Scalability features

A *scalable* application responds well under increased loads. RS can scale up and out to address the high-volume reporting requirements of large organizations. It is designed from the ground up to process reports efficiently. For example, it supports several report caching options, such as report execution caching, snapshots, and report sessions, as we discuss in chapter 7.

Reporting Services Enterprise Edition supports clustered deployment, which you can use to load-balance several RS servers on multiple machines. This allows enterprise organizations with high-scalability requirements to scale out RS and provides fault tolerance. RS performance is the subject of chapter 16.

1.2.6 Security features

RS is designed to provide a secured environment from the ground up. It offers a comprehensive security model for accessing reports that leverages Windows authentication. This model maps the user Windows account or group to a *role*, and the role describes what permissions the user has to access items in the report catalog. Report administrators can add Windows users to predefined roles or create new ones.

Once again, when the default Windows-based security model is not a good fit, you can replace it by plugging in your own custom authentication and authorization implementations in the form of custom security extensions.

To promote trustworthy computing, RS leverages the .NET code-based security to "sandbox" custom code based on configurable security policies. We discuss the RS security model in chapter 8.

1.2.7 Deployment features

Because it is server-based, RS has zero deployment requirements for integrating with client applications. For this reason, any type of client applications can target RS, not only .NET-based applications. Because you can access RS through the two most popular web protocols, HTTP-GET and Simple Object Access Protocol (SOAP), any web-capable application can be integrated with RS, regardless of the targeted platform and development language.

DEFINITIONS The Hypertext Transfer Protocol (HTTP), on which the Internet is based, comes in two flavors: HTTP-GET and HTTP-POST. While HTTP-GET passes request parameters as a part of the URL, HTTP-POST passes them as name/value pairs inside the actual message.

Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol, layered on top of HTTP, for exchanging structured and type information on the Web. In recent years, SOAP has become the industry-standard protocol for communicating with web services.

Integrating your applications with RS requires a good grasp of its architecture. The next section outlines the major RS building blocks.

1.3 RS ARCHITECTURE

An important feature of the RS architecture is that it is *service*-oriented as opposed to *object*-oriented. Don Box, a Microsoft prominent architect working on the next-generation web services, outlines the following four characteristics of a service-oriented architecture:

- *Boundaries are explicit*. Cross-application communication uses explicit messaging rather than implicit method call invocation.
- *Services are autonomous*. The lifetime of a service-oriented application is not controlled by its clients.
- *Services share schema and contract, not class*. Service-oriented applications advertise their functionality to the outside world using XML-based schemas.
- *Service compatibility is determined based on policies*. By using policies, service-oriented applications indicate which conditions must be true in order for the service to function properly.

You may have used object-oriented reporting tools in the past in which the report consumer instantiates an object instance of the report provider. A characteristic of this model is that both the report consumer and the report provider instances share the same process space. For example, to render a Microsoft Access report, you need to instantiate an object of type `Access.Application`. Then, you use OLE automation to instruct Access to open the report database and render the report.

You will probably agree that as useful and widespread as the object-oriented model is, it is subject to some well-known shortcomings. For example, both the consumer and provider are usually installed on the same machine. As a consequence, the reports hosted by the report provider are not easily accessible by geographically dispersed clients. For instance, only COM-capable clients can interface with Microsoft Access.

A second shortcoming involves application interdependencies. Object-oriented applications are typically deployed as a unit. All Microsoft Access clients, for example, need to have the Access type library installed locally in order to establish a reference to it.

To address these shortcomings, RS departs radically from the object-oriented paradigm. In terms of reporting, the RS service-oriented architecture offers two distinct advantages: (1) Administrators can centralize the report storage and management in one place, and (2) it promotes application interoperability—report consumers can request reports over standard web protocols, such as HTTP-GET and SOAP.

The RS service-oriented architecture can be better explained in the context of a three-tier application deployment view, as shown in figure 1.3.

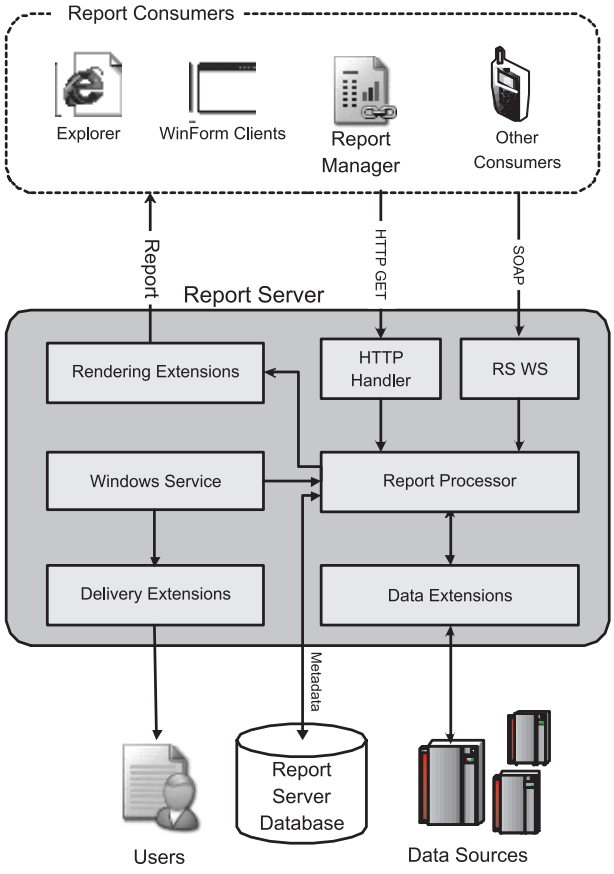


Figure 1.3
Report consumers submit report requests to the Report Server, which queries data sources to retrieve the report data and generate the report.

The RS architecture includes the following main components:

- The Report Server, whose main task is to generate reports
- The Report Server Configuration Database (the report catalog), which serves as a centralized report repository
- The Report Manager, a web-based tool for managing the report catalog and requesting reports.

Let's explain the role of each component in more detail, starting with the Report Server.

1.3.1 The Report Server

At the heart of the RS architecture is the Report Server engine. The Report Server performs the following main tasks:

- Handles the report requests sent by the report consumers. I will use the term "*report consumer*" to describe any client application that requests reports from the Report Server. Once again, this could be *any* application regardless of the language in which it was written or the platform it runs on.
- Performs all chores needed to process the report, including executing and rendering the report, as we discuss in detail shortly.
- Provides additional services, such as snapshots and report caching, authorization and security policy enforcement, session management, scheduling, and subscribed delivery.

DEFINITION We will use the term "report request" to refer to the set of input arguments that the report consumer has to pass to the Report Server to generate a report successfully. At minimum, the report request must specify the path to the report and the report name. Other arguments can be passed as report parameters, including rendering format, whether the report should include the standard toolbar, and so forth.

Looking at figure 1.3, you can see that the Report Server encompasses several components, including the Report Processor, Windows Service, and extensions. From an implementation standpoint, perhaps the best way to describe the Report Server is to say that it is implemented as a set of .NET assemblies located in the C:\Program Files\Microsoft SQL Server\MSSQL\RS\ReportServer\bin folder.

NOTE An interesting fact about the Report Server is that it is 100% written in C# code. As far as I can tell, this qualifies it as the first true .NET server. No, unfortunately the source code is not provided. Moreover, the Report Server assemblies are obfuscated to prevent reverse engineering, reuse, and abuse.

As you know, the Report Server's main role is to generate reports. To accomplish this, the server retrieves the report definition from the report catalog, combines it with data from the data source, and generates the report.

Figure 1.3 and the product documentation indicate that the Report Processor component is responsible for report processing. The implementation details of the

processor are not disclosed at the time of this writing, but most likely the majority of its functionality is encapsulated in the Microsoft.ReportingServices.Processing.dll assembly. For the remainder of this book we use the terms *Report Processor* and *Report Server* interchangeably.

Section 1.4 explains the purpose of each of the Report Server components and shows how they relate to report processing.

From an integration standpoint, perhaps the most important observation that you need to draw from figure 1.3 is that the Report Server has two web-based communication façades that expose its functionality to external clients: HTTP Handler, which accepts URL-based report requests submitted via HTTP-GET, and the Web service (shown in figure 1.3 as RS WS), which handles SOAP requests. You will see how these façades impact the report-delivery process in section 1.5.

1.3.2 The Report Server database

When you install RS, the setup program creates the Report Server database. This database is implemented as two physical SQL Server 2000 databases: The Reporting Services Configuration Database, ReportServer, hosts the report catalog and metadata. In this section, we'll take a closer look at each.

The Reporting Services Configuration Database

The Reporting Services Configuration Database, ReportServer, hosts the report catalog and metadata. As we mentioned earlier, in order for a report to be available to the end users, its report definition file must be uploaded (published) to the catalog.

If you open this database in the SQL Server Enterprise Manager, you will be able to deduce the purpose of most of its tables. For example, the Report Server Configuration Database keeps the catalog items in the Catalog table, the data source information in the Data-Source table, and so forth. Note that querying the report catalog directly is discouraged by Microsoft. Instead, the recommended way to access the report catalog is through the Report Server APIs. Microsoft also discourages you from making data changes directly to the catalog. The reason behind this is that Microsoft may change the catalog schema in the future but will maintain backward compatibility through the Report Server API.

As you may recall, RS can be deployed in a load-balanced cluster environment. In this deployment model, the Report Server database is shared among all nodes of the cluster.

The Reporting Services Temporary Database

The RS setup program also creates a second database, ReportServerTempDB, which is used by RS for caching purposes. For example, once the report is executed, the Report Server saves a copy of the report in the ReportServerTempDB database.

DEFINITION *Report caching* describes the Report Server feature of keeping the report intermediate format in the Report Server database for a certain duration.

We'll return to the topic of report caching in chapter 7.

The Adventure Works 2000 sample database

Finally, if you install the RS samples, the setup program installs a sample database called AdventureWorks2000. This database is also used by other Microsoft products, such as Commerce Server and Notification Services.

The AdventureWorks2000 database includes a much more "realistic" sales ordering database model than the SQL Server sample databases, Northwind or Pubs. You will quickly realize this by surveying the data held in the more than 60 tables. We'll work with this sample database in section 1.7, where you'll have a chance to create a report using RS.

1.3.3 The Report Manager

Implemented as an ASP.NET web application, the Report Manager performs two main tasks: report management and requests for reports. You can think of the Report Manager as an application façade that communicates with the Report Server via the Report Server APIs. From the Report Server perspective, the Report Manager is no different than any other client application.

Report management

Users familiar with SharePoint Portal Server will find the Report Manager similar to this product both in terms of user interface and purpose. As you can with SharePoint, you can use the Report Manager to create folders, upload resources, manage subscriptions, and set up security.

For example, figure 1.4 shows that I used the Report Manager to navigate to a folder AWReporter and to retrieve a list of the catalog items under this folder. You can click on a report link to run a report or access and change the report properties.

In case you're wondering where the items shown in figure 1.4 come from, we will create them in the next few chapters when we discuss the report-authoring process.

Keep in mind that in RS you work with virtual folders. Neither the folders nor the report definition files actually exist in a file system. Instead, they exist in the Report Server Database as metadata, but they appear as folders and items when you access the Report Server through the Report Manager.

Requesting reports

Sometimes, building a reporting application might be overkill. Or small companies might not have the IT resources to do so quickly or simply cannot afford the effort. In such cases, the Report Manager can be used as a reporting tool. Users can navigate to the Report Manager portal and request reports on the spot, as figure 1.5 shows.

Even better, users can use the handy toolbar, which the Report Server generates automatically, to perform various report-related tasks, including specifying parameter values for reports that take parameters (more on this in chapter 3), paging, zooming, and exporting the report to different formats.

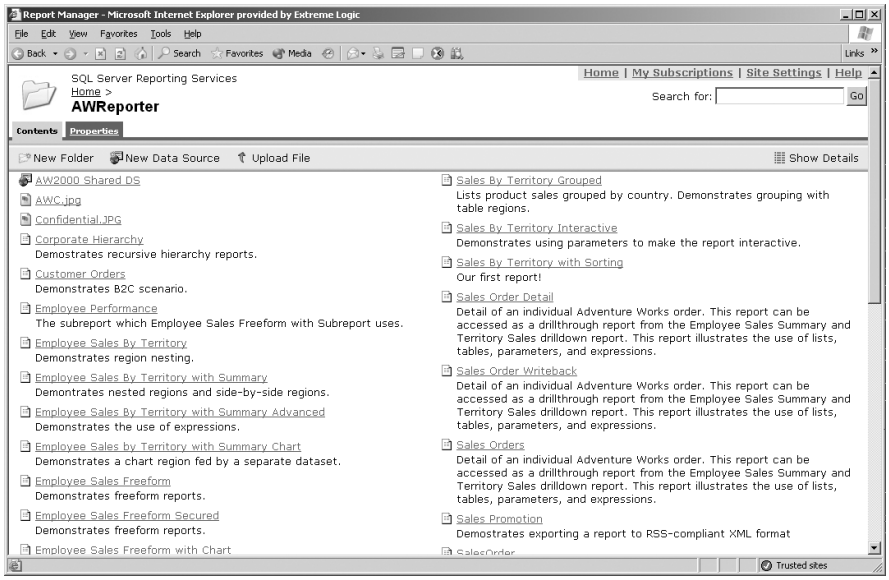


Figure 1.4 Users can use the Report Manager portal to generate or manage reports.

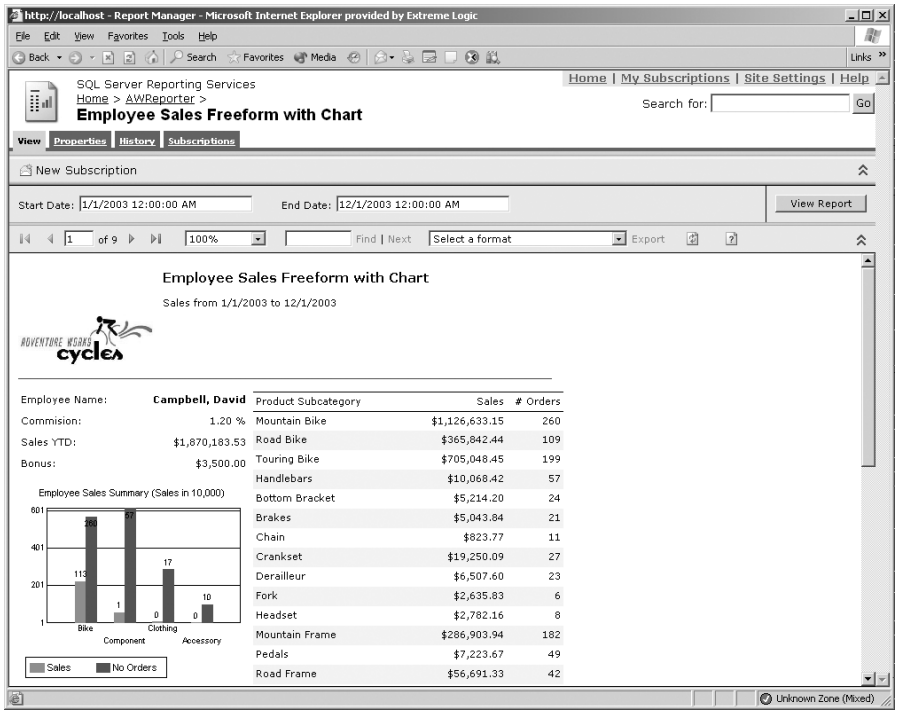


Figure 1.5 Small organizations that don't need to create report-enabled applications can use the Report Manager to request reports. This figures shows the Employee Sales Freeform with Chart report generated in HTML.

Now that we've had a 100-foot view tour of the major building blocks of RS, let's peek under its hood to see how it processes, renders, and delivers reports.

1.4 UNDERSTANDING REPORT PROCESSING

Report processing encompasses all activities performed by the Report Server to generate a report. To understand how the Report Server processes a report, let's see what happens when the report is requested on demand.

Figure 1.6 depicts what happens when a report hosted under the Report Server is requested by a report consumer. First, the consumer submits (1) a report request to the Report Server.

Once the report request is intercepted by the Report Server, it is forwarded (2) to the Report Processor. The Report Processor parses the request and retrieves (3) the report definition and metadata from the Report Server Database. The Report Processor checks whether the user is authorized to access this report. If so, the Report Processor processes the report, which involves two stages: execution and rendering.

Let's get more insight into each of these stages, starting with the execution stage.

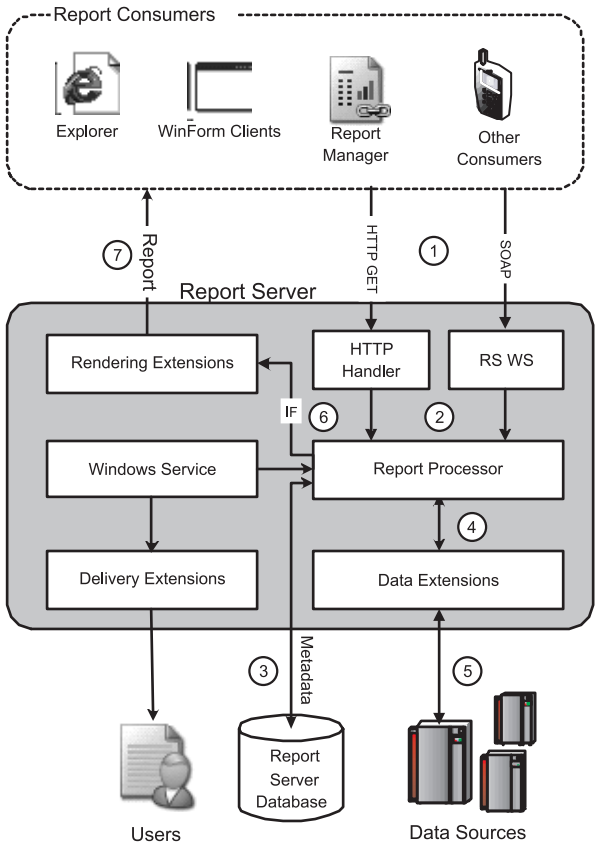


Figure 1.6
You can integrate your applications with RS by using the two web communication façades: HTTP Handler and the RS Web service.

1.4.1 Execution stage

The report execution phase starts when the Report Server begins processing the report and finishes when the report is ready for rendering. For the sake of simplicity, let's assume that the report is requested for the first time.

Generating the raw report

As we explained earlier, when the report is published, the Report Server parses its report definition (RDL), generates a .NET assembly, and saves the assembly in the catalog for the report. During the execution phase, the Report Server loads and executes the assembly. Referring back to figure 1.6, you can see that the Report Server uses a data extension (4) to query (5) the data source to retrieve the report data, combines the resulting dataset and report layout information, and produces (6) the report in a raw form, called *intermediate format* (IF).

Having the report generated in an intermediate format before it is finally rendered is beneficial in terms of performance. It allows the Report Server to reuse the same IF regardless of the requested export format. Developers who are familiar with the intermediate language (IL) code execution model in .NET can think of IF in a similar way. IL abstracts the platform on which the code executes, while IF abstracts the rendering format. For example, one report consumer can request the report in an HTML format, while another can request the same report as PDF. In either case, the Report Server already has the raw report; the only thing left is to transform it into its final presentation format. During the rendering stage, the Report Server loads the report IF and renders (7) the report in the requested format using a rendering extension.

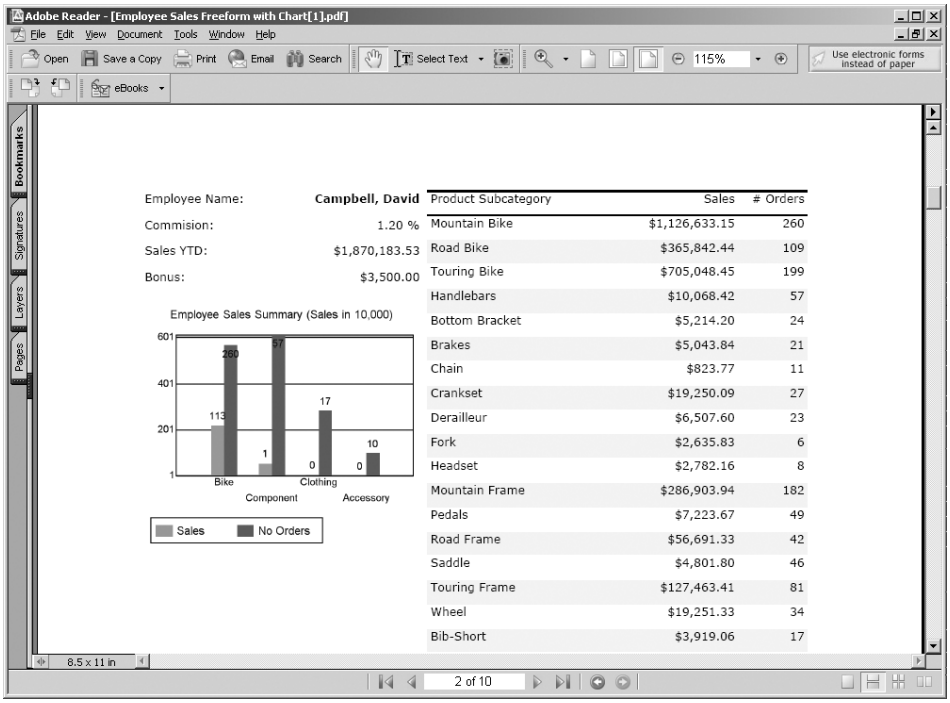
Once the report IF is generated, it is saved (cached) in the Report Server Temporary Database. Note that if the report is cached, the report execution phase may be bypassed completely for subsequent requests because the Report Server decides to use the cached IF. We will postpone discussing report caching until chapter 7.

1.4.2 Rendering stage

As shown in figure 1.6, the report-rendering stage represents the second (and last) stage in the report-processing pipeline. After the Report Server has the report IF, it renders the report in its final presentation format as per the export format requested by the user. You will be pleasantly surprised to see the plethora of natively supported formats that a report can be exported to. My favorites are HTML and PDF. For example, as figure 1.7 shows, I have loaded a report exported to a PDF file in Adobe Acrobat.

The Report Server delegates the report-rendering process to rendering extensions. RS comes with various rendering extensions that correspond to supported export formats. If the report consumer does not specify the export format explicitly, the report will be rendered in HTML 3.2 or 4.0, depending on the browser capabilities. Table 1.5 lists each out-of-the box RS supported rendering formats.

As we've said before, when the supported formats are not enough, you can write your own rendering extensions.



Product Subcategory	Sales	# Orders
Mountain Bike	$1,126,633.15	260
Road Bike	$365,842.44	109
Touring Bike	$705,048.45	199
Handlebars	$10,068.42	57
Bottom Bracket	$5,214.20	24
Brakes	$5,043.84	21
Chain	$823.77	11
Crankset	$19,250.09	27
Derailleur	$6,507.60	23
Fork	$2,635.83	6
Headset	$2,782.16	8
Mountain Frame	$286,903.94	182
Pedals	$7,223.67	49
Road Frame	$56,691.33	42
Saddle	$4,801.80	46
Touring Frame	$127,463.41	81
Wheel	$19,251.33	34
Bib-Short	$3,919.06	17

Figure 1.7 With RS you can export your reports to many formats, including Adobe Acrobat PDF. Here, I have exported this report to Adobe PDF and loaded it in Adobe Reader.

Table 1.5 Report rendering options

Rendering Extension	Description
HTML	HTML 4.0 (Internet Explorer 5.0 and above), Netscape (6.2 and above), HTML 3.2 otherwise.
HTML with Office Web Components	HTML 4.0 with Office Web Components (OWC). Charts are rendered using the OWC chart control and matrixes are rendered using the PivotTable OWC.
MHTML	MIME encapsulation of the Aggregate HTML Documents standard, which embeds resources such as images, documents, or other binary files as MIME structures within the report. This is a good option to minimize the number of round trips between the browser and server to fetch resources. MHTML is most useful for sending reports through e-mail, as we see in chapter 14.
PDF	Adobe Acrobat files.
Excel	Creates a visual representation of the report in an Excel workbook and translates Excel formulas whenever possible. Users can open the report in Excel to change it.

continued on next page

Table 1.5 Report rendering options *(continued)*

Rendering Extension	Description
XML	Creates an XML document containing the information in the report. The schema of the XML document generated is determined by the contents and layout of the report. Users can use the Data Output tab in the Report Designer to control how the elements will be rendered.
CSV	Comma-separated value file, with no formatting.
Image	Renders reports to bitmaps or metafiles, including any format that GDI+ supports: BMP, EMF, GIF, JPEG, PNG, RIFF, and WMF. By default, the image is rendered in TIFF, which can be displayed with an image viewer. Image rendering ensures the report looks the same on every client. Rendering occurs on the server; all fonts used in the report must be installed on the server.

Once the report is generated it is ready to travel to its final destination: the report user. RS gives you a lot of flexibility to distribute your reports, as you'll see in the next section.

1.5 DELIVERING REPORTS

As we mentioned earlier, RS supports both on-demand (pull) and subscribed (push) report delivery. To view a report on demand, the user explicitly requests the report from the Report Server. Alternatively, the user can choose to subscribe to a report. With this option, the report is pushed to the subscribers when the report data is refreshed or on a specified schedule.

Let's take a closer look at each delivery option.

1.5.1 On-demand delivery

One of the most important decisions you have to make when integrating RS reports in your application is how the application will access the Report Server to request reports. While in some cases the system design may dictate the integration option, occasionally the choice won't be so straightforward and you may have to carefully evaluate the application requirements to determine the best approach. We revisit the on-demand delivery options in more detail in chapter 9. For now, note that reports can be requested on-demand in two ways: URL access and the Web service.

URL-based report access

The report consumer requests a report by URL by submitting an HTTP-GET request to the Report Server. The advantages of URL access are its simplicity and better performance. In the simplest case, the consumer can embed the report URL into a hyperlink.

For example, a web-based application can have a drop-down Reports menu where each link targets a RS report. With the URL access option, the report arguments are

passed as query parameters in the report URL. For example, assuming that you have installed the sample reports included with book source code, the following URL will run the Territory Sales Crosstab sample report with the start date 3/1/2003 and an end date of 4/30/2003.

```
http://localhost/ReportServer?/AWReporter/Territory Sales Crosstab&Start-
Date=3/1/2003&EndDate=4/30/2004
```

Web service

With RS, reports can also be requested by submitting SOAP-based requests to the Report Server Web service. The main advantage of this service is that its feature set goes well beyond just report rendering. It also encompasses an extensive set of methods to manage all aspects of the Report Server, such as uploading reports, retrieving a list of resources from the report catalog, and securing RS.

You can think of the Report Server Web service as a façade to the Report Server that allows RS to be integrated with a broad array of platforms. For example, if you are building an enterprise application integration (EAI) solution, a BizTalk schedule might invoke the Web service `Render()` method, get the XML representation of the report, retrieve some data from it, and pass it on to another application. Or, if your reporting application is B2B oriented and your partner has a Web service, you can send the report results to it in XML.

In some cases, a report consumer will use a combination of both access options to integrate with RS. For example, a report consumer can use the RS Web service to find out what parameters a report takes. Then, the application presentation layer can present the parameters to the user so that the user can enter the parameter values. When the user submits the report request, the application can use URL access to send the request to the Report Server.

1.5.2 Subscribed delivery

In the "push" report delivery scenario, the reports are generated and delivered automatically by the Report Server to a delivery target. Reports can also be delivered at a scheduled time. For example, a financial institution can set up a portfolio balance report to be generated and delivered through e-mail to its customers at the end of each month.

The Report Server Windows service (ReportingServicesService.exe) works in tandem with the SQL Server Agent service to generate and deliver subscribed reports.

> **NOTE** SQL Server Agent is a component of Microsoft SQL Server, and it is responsible for running scheduled SQL Server tasks.

For example, if the report is to be generated according to a set schedule, the SQL Server Agent will create a job and move the subscription to the Subscriptions table when the time is up. The RS Windows service periodically polls the Report Configuration Database to find out whether there are any new subscription jobs. If this is

the case, the Windows service picks up the job, generates the report, and delivers it to the end users through a delivery extension.

Out of the box, RS comes with two delivery extensions: the e-mail delivery extension and the file share delivery extension. The e-mail delivery extension delivers the report via e-mail. The report can be delivered to either subscribed users (opt-in subscription) or to a data-driven list of recipients. The file share extension delivers reports to a network share. When these two options are not enough, you can write custom delivery extensions.

Note that the Report Server Windows service doesn't communicate with the Report Server through the HTTP Handler or Web service façades. Instead, because it is installed on the same machine as the Report Server, the Windows service directly loads and calls the Report Server assemblies. This is beneficial for two reasons. The first relates to availability. Even if the IIS server is down, the Windows service will still execute scheduled tasks and deliver reports to subscribers. The other reason is better performance—the web façades are completely bypassed.

Another task that the Report Services Windows service is responsible for is performing background database integrity checks, as well as other administrative tasks.

Before we see RS in action, it may be beneficial to get a good high-level understanding of the report lifecycle. This is important because the remaining chapters of this book follow an identical flow.

1.6 WHAT IS THE REPORT LIFECYCLE?

By now, you probably realize that the Report Server is a sophisticated reporting platform with a feature set that goes well beyond a desktop reporting tool. To minimize the learning curve, this book follows a logical path based on the *report lifecycle*. The report lifecycle is the process that you typically follow to work with reports, and it involves three stages: authoring, management, and delivery. Figure 1.8 depicts the report lifecycle stages.

In the report-authoring stage, you create the RDL file through the use of report-authoring tools. For example, you can use the Visual Studio .NET Report Designer to lay out the report. Recall that both report data retrieval and layout information are described in the RDL file. We'll discuss many more details of the report authoring stage in chapters 2–6.

In the report-management stage, you manage the report catalog. As you recall, the report catalog is stored in the Reporting Services Configuration Database. The report catalog keeps the report and all related items. Typical management tasks include organizing reports in folders, uploading reports, and granting users access to run reports. We'll take a closer look at report management in chapters 7 and 8.

The report-delivery stage is concerned with distributing the reports to their final destinations, including end users, printers, or archive folders. A managed report can be delivered either on-demand or pushed to the subscribed users. Report delivery is discussed in detail in chapters 9–14.

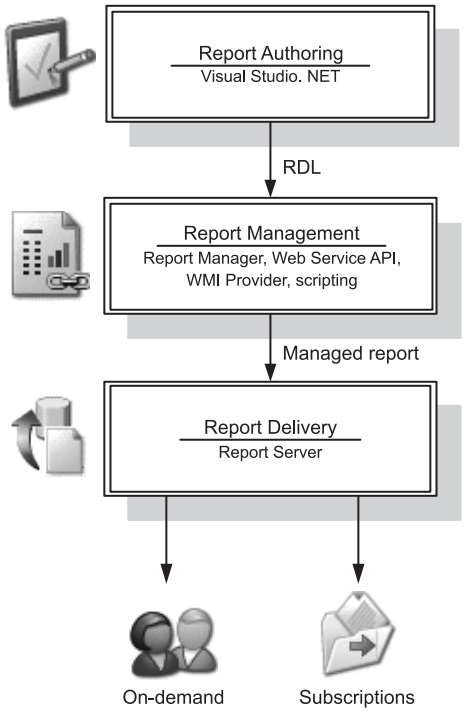


Figure 1.8
Report lifecycle phases include report authoring, management, and delivery. In the report-authoring stage, you lay out the report. In the report-management stage, you deploy and manage the report. Finally, RS gives you many ways to distribute your reports to their final destination.

Enough theory! Let's put in practice what we learned so far and get our hands on RS.

1.7 RS IN ACTION

This section has two main objectives. First, we introduce an imaginary company, Adventure Works Cycles (AWC), which we reference throughout the rest of this book. We will discuss various hypothetical reporting challenges that AWC faces and implement solutions to address them.

Second, we get our feet wet and create our first report using the Visual Studio .NET Report Wizard and the AdventureWorks2000 sample database. Granted, this is going to be a simple tabular-style report, but as simple as it is, it showcases all the phases of the report lifecycle. We also use this report in the next three chapters as a practical example to expand our knowledge about RS.

1.7.1 About the Adventure Works Reporter

Let's start with a hypothetical problem statement. You are a developer with AWC, which manufactures and sells goods to individuals and retailers. The company has enjoyed tremendous success the last few years. Sales are going up exponentially and the customer base is growing fast. Today, AWC has customers both in the United States and overseas. It has already implemented a web-based ordering online transaction processing (OLTP) system to capture sales orders online.

However, success does not come cheap. Data inaccuracy and slow decision making are among the top complaints by the sales managers. Often, data is captured and consolidated in the form of Excel spreadsheets. What is needed is a reporting system to present the company with data in a format that's both easy to understand and analyze and to allow AWC's management to discover trends and see how the company is performing. You have been designated as a lead developer for the new Adventure Works (AW) Reporter system. Fascinated by Microsoft SQL Server 2000 RS, you decide to base your reporting system on it.

NOTE In the real world, you should abstain from reporting off an OLTP database for performance reasons. As the name suggests, OLTP systems must scale to meet large transaction volumes and handle hundreds and even thousands of users. Reporting applications usually submit queries to retrieve and analyze substantial sets of data, which impose data locks on many records in the database. This can severely tax your OLTP system performance. For this reason, reporting and OLTP are usually two mutually exclusive options. A typical solution involves consolidating OLTP data and then uploading it to a data warehouse database that is optimized and designated for reporting purposes only. We discuss OLAP and data warehousing in detail in chapter 12.

1.7.2 Your first report

One crucial piece of information that the AW management would probably like to know is what the yearly products sales per territory are. With such a report in hand, managers can determine how well AW is doing in each sales region. To meet this requirement, let's create the Sales by Territory report. Figure 1.9 shows the final version of the report that we'll create in this section.

Sales By Territory

Territory	Product Category	Sales
Australia	Accessory	$8,359.74
	Bike	$773,111.46
	Clothing	$18,665.74
	Component	$96,911.11
Canada	Accessory	$32,835.84
	Bike	$2,315,632.98
	Clothing	$86,657.62
	Component	$457,317.56

Figure 1.9 Our first report is Sales by Territory.

This is just one of the many sample reports we'll design throughout the course of this book. We'll use the Sales by Territory report in subsequent chapters to demonstrate other RS features.

Table 1.6 shows the list of tasks that we need to accomplish to create the report organized by the report lifecycle phases.

Table 1.6 The task map for creating our first report

Phase	Task	Description
Authoring	Create BI project.	Create a new BI project in Visual Studio .NET.
	Create the report data source.	Use the Report Designer Data tab to configure a database connection to the AdventureWorks2000 database.
	Set the report dataset.	Define a dataset query to retrieve the report data.
	Lay out the report.	Use the Report Wizard and Report Designer to author the report.
	Test the report.	Use the Report Designer Preview tab to preview and test the report.
Management	Deploy the report.	Use Visual Studio .NET to deploy the report to the Report Server catalog.
Delivery	Ensure on-demand report delivery.	Use the Report Manager to navigate and render the report.

As you'll recall, the first phase of the report lifecycle is authoring the report.

Authoring the report

Let's develop our first report using the Report Designer. To do so, we need to create a new Visual Studio .NET Business Intelligence (BI) project.

Task: Create a Business Intelligence Project

To create a project, complete the following steps (see figure 1.10):

Step 1 Open Visual Studio .NET and choose File → New → Project.

Step 2 From Project Types, select Business Intelligence Projects.

Step 3 From Templates, select Report Project.

Step 4 In the Location field, enter **AWReporter**, specify a location, and click OK.

Step 5 Once the project is created, right-click on the AWReporter project node in the Solution Explorer window and select Properties. The Property Pages dialog box appears, as shown in figure 1.11.

Step 6 Verify that the TargetFolder setting is set to AWReporter. This specifies the folder name in the report catalog where all reports defined in the project will be deployed.

Step 7 In the TargetServerURL field, enter the Report Server URL. If RS is installed locally on your machine and you have accepted the defaults during setup, the

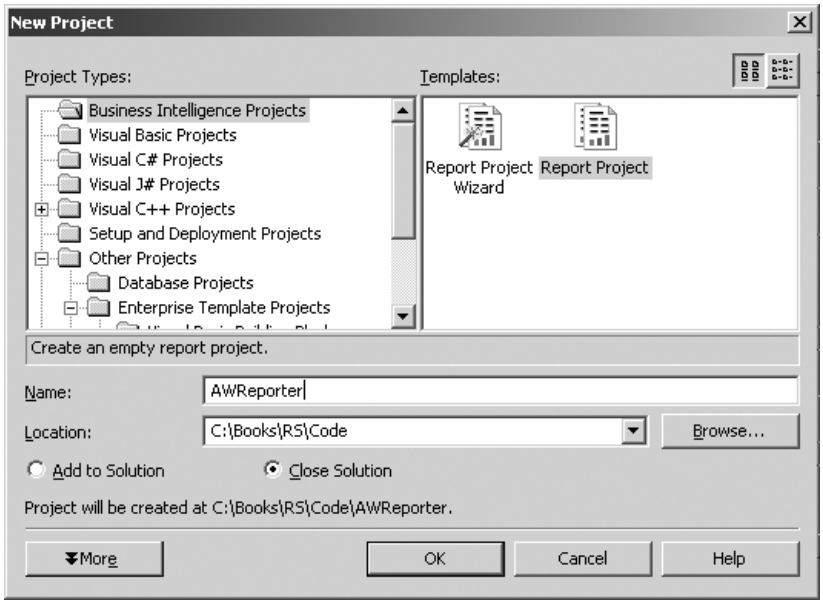


Figure 1.10 Use Visual Studio .NET to create a new BI project.

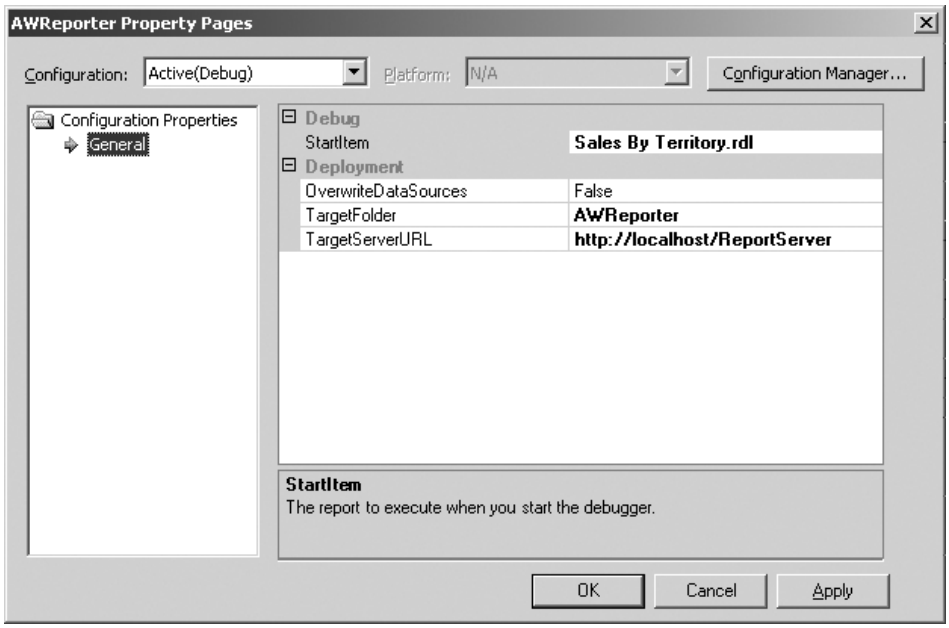


Figure 1.11 Use the report property page to set up the project properties.

URL of the Report Server should be http://localhost/ReportServer. Click OK to close the Property Pages dialog box.

Task: Create the Report Data Source

Next, we create a shared data source pointing to the AdventureWorks2000 sample database. Don't worry if the concept of a shared data source is not immediately clear. When we get to chapter 3 it will all begin to make sense.

Step 1 Right-click on the Shared Data Sources node in the Solution Explorer and choose Add New Data Source. The familiar Data Links Properties appears, as shown in figure 1.12.

Switch to the Provider tab and verify that the Microsoft OLE DB Provider for SQL Server is selected (we will be connecting to a SQL Server database). Back to the Connection tab, specify:

- The name of the SQL Server that you use to install RS. In my case, the database is installed locally, which is why the data source name is "."
- A valid username and password combination for an SQL Server account that has permissions to query the tables in the AdventureWorks2000 database. Select the Allow Saving Password check box.
- Select the AdventureWorks2000 database from the "Select the database on the server" drop-down list. Test the connection by clicking the Test Connection button. If all is well, click OK.

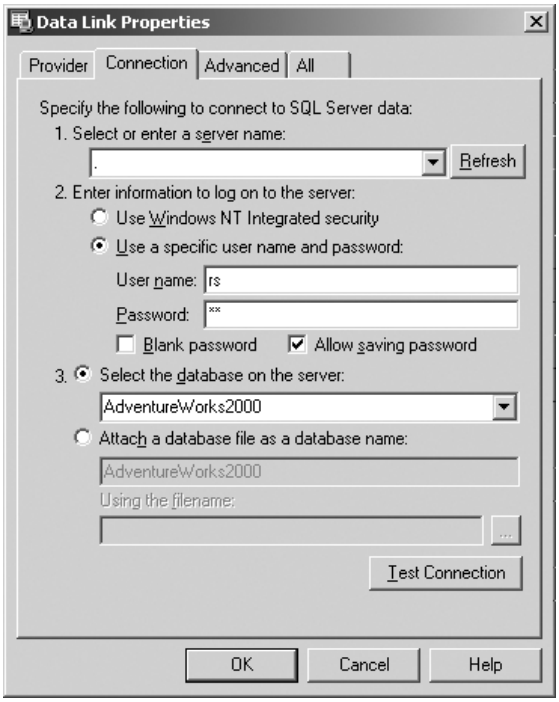


Figure 1.12
Use the Data Link Properties dialog box to establish to set up a data source pointing to the AdventureWorks2000 database.

Step 2 By default, RS names the data source with the same name as the database. Since we are going to use this data source for most of the sample reports in this book, let's make the name more descriptive.

Double-click on the AdventureWorks2000.rds file. The Shared Data Source dialog box appears, as shown in figure 1.13.

Change the Name property of the data source to AW2000 Shared DS and click OK. Optionally, in the Solution Explorer rename the data source file to AW2000 Shared DS.rds.

Now it's time to author the report. We'll use the handy Report Wizard to save some time.

Task: Set the Report Dataset

Step 1 Right-click on the Reports node in the Solution Explorer and choose Add New Report.

Step 2 On the Report Wizard welcome screen, click Next.

Step 3 On the Select the Data Source screen, make sure that the Shared Data Source radio option is selected and that AW2000 Shared DS appears in the Shared Data Source drop-down list. Click Next.

Step 4 In the Design the Query screen, click the Edit button. The familiar query designer window appears.

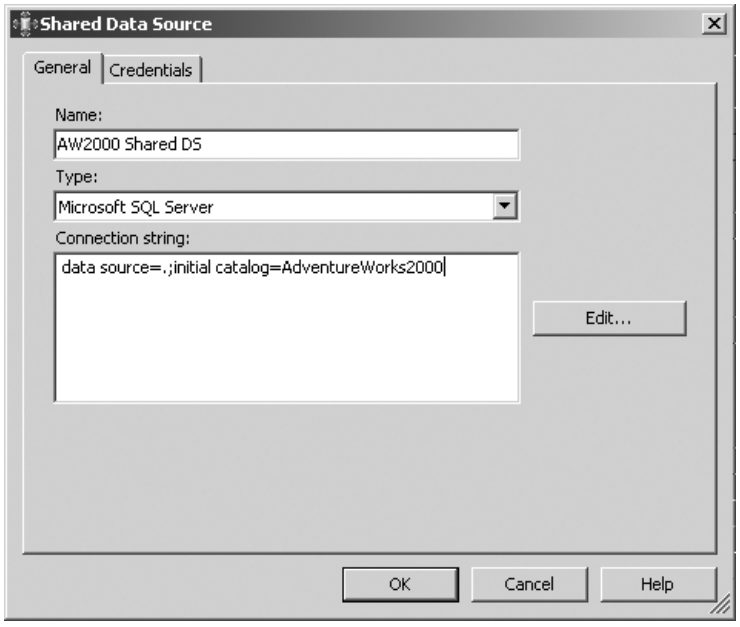


Figure 1.13 Setting up the shared data source to AdventureWorks2000 database.

Step 5 Enter the following SQL statement in the query pane:

```
SELECT        ST.Name AS Territory, PC.ProductCategoryID,
              PC.Name AS ProductCategory,
              SUM(SOD.UnitPrice * SOD.OrderQty) AS Sales
FROM          SalesOrderDetail SOD
INNER JOIN    Product P ON SOD.ProductID = P.ProductID
INNER JOIN    SalesOrderHeader SOH ON
              SOD.SalesOrderID = SOH.SalesOrderID
INNER JOIN    SalesTerritory ST ON
              SOH.TerritoryID = ST.TerritoryID
INNER JOIN    ProductSubCategory PSC ON
              P.ProductSubCategoryID = PSC.ProductSubCategoryID
INNER JOIN    ProductCategory PC ON PSC.ProductCategoryID =
              PC.ProductCategoryID
WHERE         DATEPART(YY, SOH.OrderDate) = DATEPART(yy, GETDATE())
GROUP BY      ST.Name, PC.Name, PC.ProductCategoryID
ORDER BY      ST.Name, PC.Name
```

This query retrieves the product sales orders grouped by territory and product category. The AW database groups products in subcategories, which are then rolled up to product categories. For the purposes of this report, we summarize the sales data by product categories since this represents the most consolidated level in the product hierarchy, which is exactly what upper management is interested in seeing. The sales amount is retrieved from the SalesOrderDetail table. In addition, the query filters the orders created for the current year. In chapter 3, we'll make the report parameter driven by allowing the user to pass an arbitrary date. At this point, click Next.

Task: Lay Out the Report

To lay out the report, perform the following steps:

Step 1 On the Select the Report Type screen, leave the report type set to Tabular. Click Next.

Step 2 On the Design the Table screen, select all fields except ProductCategoryID and click Details so the fields appear in the report details section, as shown in figure 1.14. Click Next.

Step 3 On the Choose the Table Style screen, click Corporate, the click Next.

Step 4 Finally, on the Completing the Report Wizard screen, enter **Sales by Territory** as the name of the report. Click Finish, and we're done!

Visual Studio displays the Report Designer with the Layout tab selected, as shown in figure 1.15.

The integration with Visual Studio.NET Report Designer allows you to easily preview and test your reports without leaving the Visual Studio .NET IDE.

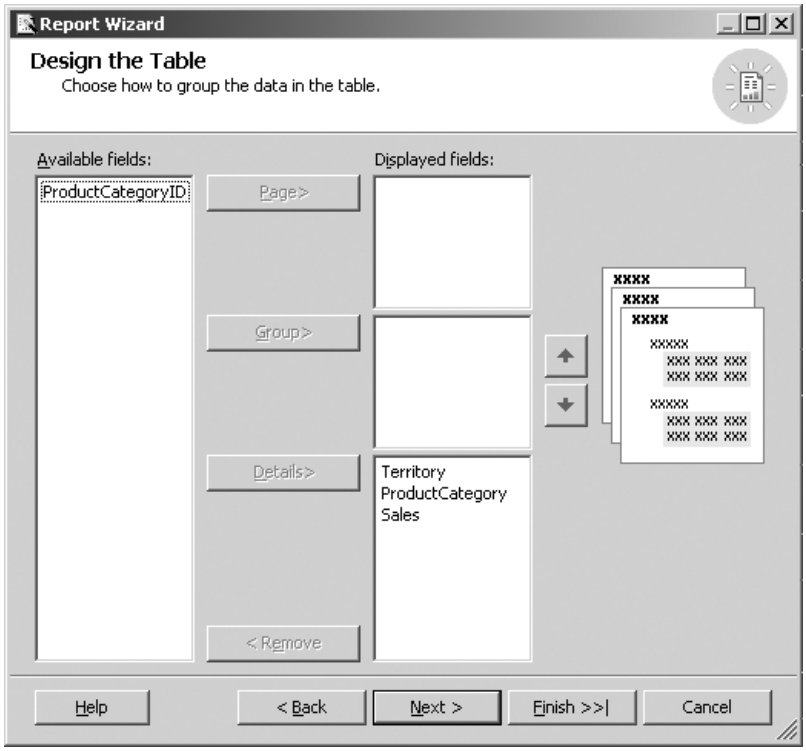


Figure 1.14 In the Design the Table step, you choose which fields will appear on the report and how data will be grouped.

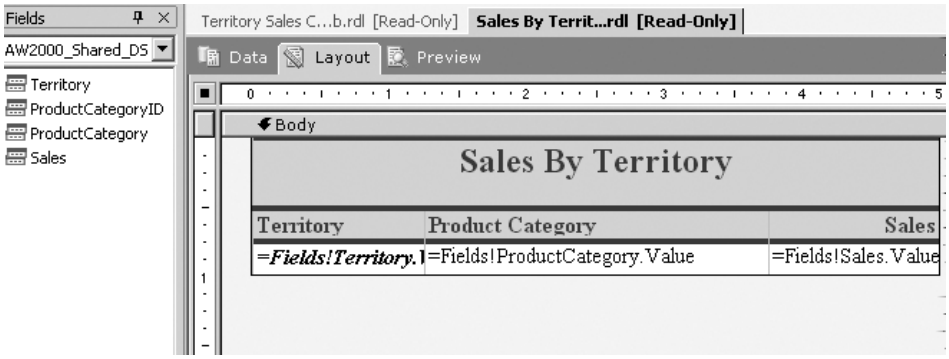


Figure 1.15 Use the Report Designer Layout tab to lay out your report.

Task: Test the Report

Let's make some cosmetic changes to enhance our report.

Step 1 Click on the Report Designer Preview tab to see the HTML representation of the report. Notice the report toolbar at the top, which allows you to zoom, print, and save the report in different formats. The Sales field needs some formatting work.

Step 2 Click the Layout tab again to go back to design mode.

Step 3 Right-click on the Sales text box and choose Properties. Specify the format settings, as shown in figure 1.16.

Click OK to close the Textbox Properties dialog box.

Step 4 Increase the width of the Territory and Product Category columns; stretch them out as far as there is space within the report width.

Step 5 Right-click again on the Territory text box and go to the field properties.

Step 6 Click the Advanced button and, in the Font tab, change the font weight to bold and style to Italic. Click OK.

Step 7 Back to the Textbox Properties dialog box, hide the repeating territory names by selecting the Hide Duplicates check box, as shown in figure 1.17.

Preview the report again. Now it should look like the report shown in figure 1.9. Still not very pleasing to the eye, but not bad for a few minutes of work!

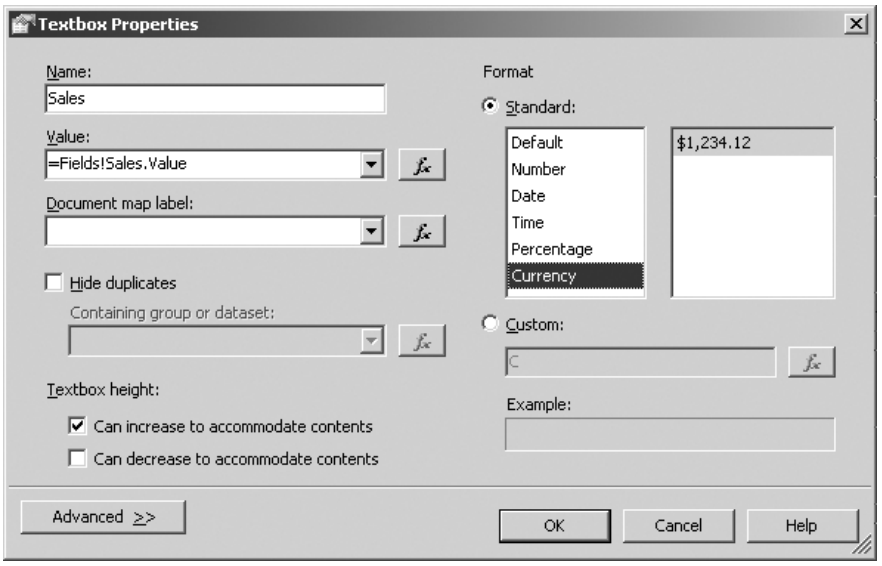


Figure 1.16 Use the Textbox properties page to set up format settings.

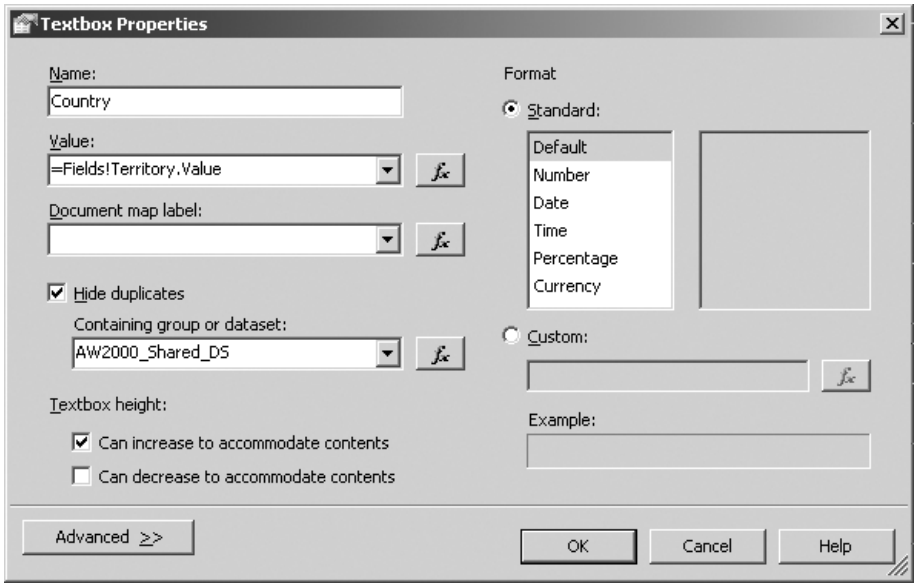


Figure 1.17 Select the Hide Duplicates check box to hide the territory name duplicates.

Report management

Once you are satisfied with the report, you will probably need to deploy it to make it available to all users. This is a report management task that you can accomplish by using the Report Manager. However, if your Windows account has local administrator rights on the computer where the Report Server is installed, you can deploy the report straight from within Visual Studio .NET. Let's do just that.

Task: Deploy the Report

Step 1 Save your changes.

Step 2 From the Solution Explorer, select the Sales by Territory.rdl node, right-click, and select View Code. Visual Studio .NET shows you the report definition of the report. Note that the report RDL includes the report query and layout information. Since we chose to create a shared data source, the data source information is not included in the report RDL.

Step 3 In the Solution Explorer, right-click on Sales by Territory.rdl and choose Deploy. This compiles the report and uploads the report to the report catalog.

Report delivery

Once the report has been promoted to a managed report, it can be delivered to your end users. Let's see how users can request the report on-demand by using the Report Manager as a quick-and-easy report-delivery tool.

Task: On-Demand Report Delivery

Step 1 Open the browser and navigate to the Report Manager URL, which by default is http://<reportservername>/reports. Notice that below the Report Manager Home folder there is a new folder, AWReporter, and that its name matches the TargetFolder setting you specified in the report project settings.

Step 2 Click on the AWReporter folder link to see its content. You should find the AW2000 Shared DS data source and Sales by Territory report links.

Step 3 Click on the Sales by Territory report link to request the report with the Report Manager.

As you can see, authoring, managing, and delivering reports with RS is straightforward. At this point, you may decide to compare RS at a high level with other reporting tools you've used in the past. The next section discusses how RS stacks up against the competition.

1.8 EVALUATING RS

By the time you read this book, comparison charts will probably be available from Microsoft and other sources to show how RS compares with other popular reporting tools. For example, the Resource section at the end of this chapter lists a link to a detailed feature comparison document between RS and Crystal Reports.

Based on my experience with integrating applications with third-party reporting packages, I think that as a first iteration, RS is surprisingly feature-rich. My favorite top ten features, where I believe RS excels, are as follows:

1. Natively exposed as a Web service—The RS reports are widely accessible, and you don't have to do anything special to publish your reports as web services because they are hosted under the Report Server, which provides a web service façade.
2. Support of plethora of export formats—You may be delighted to learn that the ability to export reports to PDF and Excel is provided out the box. In addition, reports can be delivered in many other popular formats, including web formats (HTML), popular image formats (such as TIFF and JPEG), and data formats (Excel, XML, CSV).
3. On-demand and subscribed report delivery—Another huge plus is the subscribed report delivery option, which allows developers to implement opt-in report features in their applications.
4. Documented report definition format—Developers can create reports to be published to the Report Server using Microsoft or third-party design tools that support the RS XML RDL.
5. .NET Framework integration—In the extensibility area, you'll appreciate the fact that you are not locked from a programmability standpoint. As we mentioned earlier, when built-in features are not enough, you can reach out and

borrow from the power of the .NET Framework by integrating your reports with .NET code. In addition, the Report Services programming model is 100 percent .NET-based.

6. Extensible architecture—The RS architecture is fully extensible and allows developers to plug in their own security, data, delivery, and rendering extensions.
7. Zero deployment—Thanks to its service-oriented architecture, RS has no client footprint and offers true zero deployment for all application types.
8. Scalability—RS can scale better, since it is designed from the ground up to scale in web farm environments.
9. Visual Studio .NET integration—Report authors will enjoy the familiar IDE environment when designing and testing reports.
10. Cost—From a cost perspective, it is hard to beat the bundled with the SQL Server RS pricing model, especially if you compare it with the five-digit price tag of third-party reporting tools.

Of course, nothing is perfect, and Report Services has its own shortcomings, some of which I would like to mention here. As a .NET developer, I would like to see a future version of RS bring a tighter integration with Visual Studio .NET. Ideally, working with BI projects should not be much different than working with .NET code projects, for example, Windows Forms. In the future, I would expect RS to evolve and add the following features:

- Allow developers to add code-behind files to their reports.
- Instead of Visual Basic .NET only, support all .NET-compatible languages for writing expressions and report-specific code.
- Use the Visual Studio .NET code editor instead of the Notepad-like Custom Code Editor.
- Support events; currently, developers cannot write event handlers to respond to runtime conditions. Microsoft Access, for example, has been enjoying a report object model with events since its first release. Since the RS generation process is not event-driven, the only option for implementing runtime code customization with Report Services is to use expressions.
- Include more flexible object model, for example, creating report elements dynamically, and referencing and changing report items from custom code.
- Convert reports from reporting tools other than Microsoft Access.

I hope that the above shortcomings will be addressed in future releases of RS to make this tool an even more compelling choice for enterprise reporting.

1.9 SUMMARY

This chapter took you on a whirlwind tour of the RS platform. We've discussed its role in the Microsoft BI initiative, as well as its features and high-level architecture. You have even had a chance to use RS and create a simple report based on the AdventureWorks2000 sample database. Now that you have a good high-level understanding of its features, you can begin using RS to report-enable your own applications.

By now, you should understand the major components of RS and their role in the report lifecycle. In addition, you should see the advantages that the service-oriented and web-enabled RS architecture has to offer.

Perhaps most important, you should be familiar with the three stages of the report lifecycle: report authoring, management, and delivery. The remaining chapters explore each of these stages in this order. In the next chapter, we discuss different ways to create RS reports.

1.10 RESOURCES

Microsoft RS website
(www.microsoft.com/sql/reporting/)—First stop for the latest on RS.

Microsoft Business Intelligence Platform website
(www.microsoft.com/sql/evaluation/BI/default.asp) —The Microsoft BI portal home page.

A feature comparison between RS vs. Crystal Reports
(http://certia.ramblainf.com/pdf/RSvsBO_En_v1.pdf)—Ceria's Business Intelligence Team has developed a detailed feature comparison document that outlines how Microsoft RS stacks against Crystal Enterprise.

A Guide to Developing and Running Connected Systems with Indigo
(http://msdn.microsoft.com/msdnmag/issues/04/01/Indigo/)—In section 1.3 I emphasized the role of the RS service-oriented programming model. Read Don Box's article for more information about SOA.

Microsoft Reporting Services IN ACTION

Teo Lachev

Business reporting is a lifeline of business, so a better reporting environment is a big deal. With a sophisticated, modern tool like Microsoft Reporting Services, you can report-enable any type of application, regardless of its targeted platform or development language.

Written for information workers, system administrators, and developers, this book is a detailed and practical guide to the functionality provided by Reporting Services. It systematically shows off many powerful RS features by leading you through a dizzying variety of possible uses. Following a typical report lifecycle, the book shows you how to create, manage, and deliver RS reports.

In the first half, you will master the skills you need to create reports. System administrators will learn the ropes of managing and securing the report environment. The second half of the book teaches developers the techniques they need to integrate RS with their WinForm or web-based applications. It exercises RS through a wide variety of real-world scenarios—one of this book's strengths are its many useful examples.

What's Inside

- Extend RS with custom code
- Expose reports as RSS feeds
- Implement dynamic reports with Office Web Components
- Create reports off ADO.NET datasets
- Deliver reports to Web Services
- Customize RS security
- Evaluate RS performance and capacity
- and much more

A technology consultant with the Enterprise Application Services practice of Hewlett-Packard, **Teo Lachev** has more than 11 years' experience designing and developing Microsoft-centric solutions. He is a Microsoft Certified Solution Developer and a Microsoft Certified Trainer. Teo lives in Atlanta, GA.

MANNING $39.95 US/$55.95 Canada