Microsoft TechNet

Microsoft SQL Server 9.0 Technical Articles

Reporting Services: Using XML and Web Service Data Sources

SQL Server Technical Article

Jonathan Heide
Microsoft Corporation

September 2006
Revised May 2007

Applies to:
Microsoft SQL Server 2005 Reporting Services

Summary: This white paper consolidates general information, best practices, and tips for designing Microsoft SQL Server Reporting Services (SSRS) reports. It is intended to provide a starting point for design questions and an overview of some of the capabilities of Reporting Services. (12 printed pages)

Download the source document of this white paper in Microsoft Word format [http://download.microsoft.com/download/f/1/c/f1cf7b8d-7fb9-4b71-a658-e748e67f9eba/reporting_services_xml_data_sources.doc] .

Contents

About This Document

New to Microsoft SQL Server 2005 Reporting Services (SSRS) is integrating Report Services directly with XML data sources and Web services. This white paper provides general information and tips for designing reports using these sources. It is intended to provide a starting point and overview of the available features, in addition to outlining many common scenarios. The intended audience is more experienced report authors who want to extend and integrate reporting with more services-oriented environments.

Other Sources of Information

This white paper is not intended to be an exhaustive source of information about Reporting Services. For detailed information, see the Reporting Services Web site [http://go.microsoft.com/fwlink/?LinkID=19862] .

Product Versions

This white paper is written for SQL Server 2005 Reporting Services; many features addressed are not relevant to earlier versions of Reporting Services.

Introduction

Reporting Services is a comprehensive reporting tool that integrates with a diverse set of data sources. New to Reporting Services is the ability to query directly from XML data sources and Web services. This is obtained by using the XML data provider. The XML data provider flattens the XML structure into a data set that can be accessed by the reporting engine. The flattening and integration with diverse Web services, Web pages, and arbitrary XML documents is powerful, but sometimes confusing.

This document outlines common problems and solutions encountered by report authors. It starts with a description of the XML data provider and supported data sources. Next, the document describes the features and syntax of query language, followed by an explanation of the auto-detection algorithm. It concludes with examples showing how to create reports using the XML data provider.

XML Data Provider

Reporting Services supports an extensible set of data providers, including Microsoft .NET Framework, ODBC, and OLE DB. The XML data provider is a .NET data provider that is provided with Reporting Services. This section describes behavior and query syntax unique to the provider.

XML Data Sources

There are several ways that you can use the XML data provider to access XML content. These include the following:

XML Embedded Within the Query

The XML content can be embedded directly within the query. This lets you use the expression capabilities within the processing engine to build queries and data dynamically within the report. This can be used for retrieving XML data directly from an external data source, passing it using parameters, and embedding it within the query.

XML Using URL

XML content can also be accessed directly from a URL. Notice that only the HTTP protocol is supported and the request uses the GET method. However, if parameters are specified, the POST method will be used.

XML Using Web Services

To support service-oriented architectures, the XML data provider can query Web services directly by parsing the XML structure of the SOAP response directly. This requires knowledge of the Web service structure, which includes the namespace, method, SOAP Action, parameters, and schema of the response body.

When you access these data sources, you must specify what data is needed for retrieval. You can use the XML Data Provider Query language outlined in the next section to do this.

XML Data Provider Query Language

The query language supported through the XML data provider resembles the XML Path language (XPATH). However, there are both syntactical and behavioral differences between the two languages. Perhaps the most notable difference is the lack of querying and filtering support. For a complete description of the query syntax, see SQL Server Books Online [http://go.microsoft.com/fwlink/?LinkID=50478] .

Example 1

Copy Code

```
<Customers xmlns:ord="http://customer_order_schema"
xmlns:ret="http://customer_return_schema">
    <Customer ID="1">
        <Name>Bob</Name>
        <ord:Orders>
            <ord:Order ID="1" Qty="6" Date="2001-01-01T01:01">Chair</ord:Order>
            <ord:Order ID="2" Qty="1" Date="2001-01-02T01:01">Table</ord:Order>
        </ ord:Orders>
        <ret:Returns>
            <ret:Return ID="1" Qty="2" Date="2001-03-01T01:01">Chair</ret:Return>
        </Returns>
    </Customer>
```

```
</Customers>
```

Element Path

The element path defines a structure by which the XML Data Provider will flatten the data. Auto-derivation will use *the first repeating pattern* in the document. For more information, see Auto-Detection of XML Structure, later in this white paper. If you require a different structure, you can specify the path of the repeating pattern, starting from the root node. This will define the rows returned from the query. For example, if you use the XML schema provided in Example 1:

Copy Code

```
Customers/Customer/Returns
```

Specific values can be specified at each node by enclosing them in braces. This will define the columns returned from the query. Notice that XML attributes can be accessed by specifying the "@" symbol plus the name. For example:

Copy Code

```
Customers/Customer{@ID}/Returns/Return{@ID}
```

Similarly, to reference the value of the node directly, specify the "@" symbol without a name.

Namespaces

If the query requires use of specific XML namespaces, they can be specified within the element path. The following example shows how a specific XML namespace can be specified:

Copy Code

```
<Query xmlns:ord="http://customer_order_schema"
xmlns:ret="http://customer_return_schema">
    <ElementPath>
        Customers/Customer/ret:Returns
    </ElementPath>
</Query>
```

The name of the namespace should be included on the query element and then prefixed on the particular node within the Element Path. If your data and query are not namespace-sensitive, you can use the **IgnoreNamespaces** attribute on the query element. You can then create queries that do not require the namespace to be specified. The default for **IgnoreNamespaces** is false.

Type Casting

By default, as items are returned as strings, you can select to cast fields specifically within the query. You can do this by specifying the type in parentheses after the element. For example:

Copy Code

```
Customers/Customer{@ID(Integer)}/Returns/Return{@Date(Date)}
```

The following types are supported:

- String
- Integer

- Boolean
- Float, Decimal
- Date
- XML

Date format uses ISO 8601: **YYYY-MM-DD[THH:MM:SS[.S][Z|SHH[:MM]]]**.

Encoding

To support embedded documents and varied Web services, the ability to decode HTML and Base64 encoding is supported in the XML data provider. Decoding can be specified at any node in the element path by specifying **(Base64Encoded)** or **(HTMLEncoded)**. When a section of XML is decoded, it is treated as a nested set of elements and can be queried as usual. For example, assuming everything that is contained within the Customers node is Base64-encoded:

Copy Code

```
Customers(Base64Encoded)/Customer/Returns/Return{@Date(Date)}
```

Fields that contain complete encoded XML documents are also supported. When decoded, they will be treated as a set of nested elements.

Auto-Detection of XML Structure

Report authors who do not have knowledge of the source XML schema can use the auto-detection feature of the XML data provider. This can be specified in the query using the "*" syntax in the Element Path. If the Element Path is omitted, or an empty string is specified, the data provider will detect the XML schema automatically.

Auto-detection functions by parsing the source XML for the first repeating pattern. This forms the schema defining how the XML will be flattened.

For example, the following XML structure yields a flattened data set.

Copy Code

```
<Customers>
    <Customer ID="1">
        <Name>Bob</Name>
        <Orders>
            <Order ID="1" Qty="6">Chair</Order>
            <Order ID="2" Qty="1">Table</Order>
        </Orders>
        <Returns>
            <Return ID="1" Qty="2">Chair</Return>
        </Returns>
    </Customer>
</Customers>
```

The flattened data set contains the following.

Table 1

Order	**Qty**	**ID**	**Name**	**Customer.ID**
Chair	6	1	Bob	1
Table	1	2	Bob	1

> **Note** Queries that contain name collisions will have the names prefixed with the name of the nodes that contain them. Here, ID is prefixed with Customer, differentiating "Customer.ID" from "Order ID".

You can combine partial paths with auto-detection for nodes at the same level. However, auto-detection will not parse nested structures under that level. For example:

Copy Code

```
Customers/Customer/Returns
```

Would auto-detect values from **<Returns>**, because it is a detail node. If the following is specified:

Copy Code

```
Customers/Customer
```

<Returns> would not be included in auto-detection, because they do not occur at the same level as Customer.

The query will also return any values included along the element path. This includes values from the Customer and Customers nodes. If you do not want values included from these nodes, you can add empty {} column definitions at each node:

Copy Code

```
Customers{}/Customer{}/Returns
```

Multiple parent-child hierarchies are not supported. In this example, Customer has both Orders and Returns. The provider may return only one set. Because the Orders hierarchy is specified first, auto-derivation will resolve it as the skeleton.

> **Note** Detail nodes not directly within the hierarchy will be ignored by auto-derivation if they appear after the first instance of the continuation of the hierarchy. In our example, the **<Name>** detail element appears before the hierarchy **<Orders>**. The auto-derivation will parse the structure in order and add the detail element. If the **<Name>** detail element were to appear after the **<Orders>** element, it would not be included.

Examples

This section will step through three examples using the XML data provider. The first example will demonstrate how to create a report that queries a Web service. The second example will demonstrate how to supply XML data to a report by using report parameters. The third example will expand upon the second, to query XML directly from a data source such as SQL Server.

Querying a Web Service

As described earlier, you can query Web services through the XML data provider. This section describes how to create a report that queries the Report Server Web service to gather a list of items that are contained in the root folder. For more information, see the Report Server Web service [http://go.microsoft.com/fwlink/?LinkId=71399] library in SQL Server Books Online.

Assumptions: This example assumes that you have SQL Server 2005 Report Services installed, which includes Report Designer. For more information about how to install Reporting Services and use Report Designer, see SQL Server Books Online [http://go.microsoft.com/fwlink/?LinkId=71400] .

To query a Web service

1. In Report Designer, create a new Report Server Project.
2. In the new project, create a new blank report.
3. In the **Data** tab, on the **Dataset** menu, select **New Dataset**.

   This will bring up the Data Source window, to create a new data source.

4. Name your data source, and select **XML** in the **Type** drop-down list.
5. Enter the following string in the **Connection String** box:

   http://<*Server Name*>/ReportServer/ReportService2005.asmx

6. In the **Credentials** tab, select **Use Windows Authentication (Integrated Security)**.
7. Determine the Web service namespace and method name.

   As we are using the Report Server Management Web service, the method name is "ListChildren" and the namespace is as follows:

   http://schemas.microsoft.com/sqlserver/2005/06/30/reporting/reportingservices

   Because the SOAP Action is auto-derived by appending the method name to the namespace, you do not have to specify it. However, if the Web service does not use this structure, you must specify the SOAP Action directly in the query.

   If you are using auto-derivation, be aware that namespaces with a trailing slash will append **Namespace/** with **/MethodName**, creating a double slash. You can specify both the **MethodName** and **SoapAction** explicitly in this case.

   Conversely, if you specify **SoapAction**, only the **MethodName** and **Namespace** will be parsed from the **SoapAction**. Slashes will be removed in this case.

8. Determine Web service method parameters.

   Look in the Web service documentation or the Web Services Description Language (WSDL) file directly for parameters expected by the method. Notice that parameters are case-sensitive. For our example, one parameter—the item name—is needed. In this example, we specify the root of the Report Server catalog "/" for the value.

9. Construct the query.

   Enter the following query into the Query Designer and execute the query by selecting the exclamation (**!**) button on the toolbar.

Copy Code

```
<Query>
   <Method Name="ListChildren"
Namespace="http://schemas.microsoft.com/sqlserver/2005/06/30/reporting/reportingse
rvices">
   <Parameters>
       <Parameter Name="Item">
           <DefaultValue>/</DefaultValue>
       </Parameter>
   </Parameters>
   </Method>
   <ElementPath IgnoreNamespaces="true">*</ElementPath>
</Query>
```

10. Construct the Element Path.

    Notice that the auto-derived element path may not contain the data we want. We must construct the element path accordingly. To do this, we need an understanding of the XML schema. You can do this in one of the following ways:

    - Look in the Web service documentation.
    - Review the WSDL file directly for the structure.
    - Run the following query against the WSDL file. This will use the XML data provider to parse the WSDL file, return information directly through the data set, and expose it in the designer.

      Copy Code

      ```
      <Query>
          <ElementPath IgnoreNamespaces="True">

      definitions{}/types{}/schema{}/element{@name}/complexType{}/sequence/element
      {@name,@type, complexType{sequence}}
          </ElementPath>
      </Query>
      ```

      When you run this directly against the Reporting Service WSDL, you get the following result:

      http://<*Server Name*>/ReportServer/ReportService2005.asmx

      Your report server will be substituted for **<*Server Name*>**, and the returned data will indicate the structure.

    When you have the structure, you can then construct the following element path. This path provides the name of each catalog item and casts it to a string.

    Copy Code

    ```
    <ElementPath IgnoreNamespaces="true">
        ListChildrenResponse/CatalogItems/CatalogItem{Name(string)}
    </ElementPath>
    ```

    Note We used **IgnoreNamespaces=true**; if you require an explicit namespace, you must specify it directly in the query. For more information, see the XML Data Provider Query Language section, earlier in this white paper.

11. Construct your report.

    Now that you have your data source created, build your report as usual. By executing the query in the **Data** tab, the designer will auto-generate the fields for use in the report layout.

Specifying XML Data Using Report Parameters

If you want to pass parameters into the XML Web service, you can pass them in as a report parameter and then pass the value to a dataset parameter. You can use the following steps to add a report parameter and a dataset parameter. For the rest to work, you will need to have already created a data source that queries a Web service that takes parameters. This example is based on the query sample from the preceding example in this article. The query that was used is included here.

Copy Code

```
<Query>
   <Method Name="ListChildren"
Namespace="http://schemas.microsoft.com/sqlserver/2005/06/30/reporting/reportingservi
ces">
   <Parameters>
       <Parameter Name="Item">
           <DefaultValue>/</DefaultValue>
       </Parameter>
   </Parameters>
   </Method>
   <ElementPath IgnoreNamespaces="true">*</ElementPath>
</Query>
```

Add a report parameter with the name of the parameter(s) in the Web service call.

To add a report parameter

1. Make sure the **Layout** tab is selected in Report Designer.
2. On the **Report** menu, click **Report Parameters**.
3. Click **Add**.
4. Enter a name for the parameter.
5. Select the **Data Type**, **Prompt**, and **Report** properties.
6. For this example, leave **Non-queried** selected.
7. Select **Non-queried** for **Default Values**.
8. In the **Non-Queried** field, enter **/Data Sources** for the value.

   Note **/Data Sources** can be replaced with a valid, existing folder name in Report Manager, if the Data Sources folder does not exist for you.

9. Click **OK**.

Add a parameter to your dataset that will pull in the value from the report parameter.

To add a dataset parameter

1. On the **Data** tab of your report, select the dataset to which you want to add the parameter.
2. Click the ellipsis (**...**) icon next to the drop-down list.
3. Click the **Parameters** tab.
4. In the **Name** column, enter the name of the parameter that the Web service accepts. In this case, it will be **Item**.
5. In the **Value** column, enter "**=Parameters!ReportParm1.Value**" (replace **ReportParm1** with the name you specified in step 4 of the previous procedure).
6. Click **OK**.
7. Execute the dataset by clicking the exclamation (**!**) button, or preview your report passing in the value for your report parameter.

Querying XML Data from SQL Server

Building upon our previous example, report authors can query XML data directly from a data source such as SQL Server. Because the XML data provider does not support direct querying from a data source, you can use a report dataset to query the XML data and pass it on to a subreport using report parameters.

Assumptions: This example assumes that you have Report Services installed, which includes Report

Designer, and have completed the previous example. It also assumes that you have the Adventure Works sample database installed. For more information about how to install and use Reporting Services, see Reporting Services [http://go.microsoft.com/fwlink/?LinkId=71400] in SQL Server Books Online.

To query XML data from SQL Server

1. In the same project you created in the previous example, create a new blank report.
2. In the **Data** tab, from the **Dataset** menu, select **New Dataset**.
3. In the **Data Source** dialog box, create a new SQL data source and set the connection to the local instance of SQL Server. Under **Query String**, enter the following query string, substituting your local server name for **<*Server Name*>**:

   Copy Code

   ```
   Data Source=<Server Name>; Initial Catalog="Adventure Works"
   ```

4. Click **OK**.
5. Enter the following query in the **Data** tab.

   Copy Code

   ```
   DECLARE @x xml
   SET @x = (select top 10 [Name] as Product from Production.Product for xml auto,
   root)SELECT @x as Product
   ```

6. To add a subreport, on the **Layout** tab, drag the subreport item from the toolbox to the **Layout** pane. Right-click the subreport and select **Properties**.
7. In the **Subreport** dialog box, on the **Subreport** menu, select the report you created in the previous example.
8. To add parameter to subreport, select the **Parameters** tab, and then set the subreport **XMLData** parameter equal to **=First(Fields!Product.Value)**. Notice that we must wrap the field reference in an aggregation expression, because it does not occur within the repeating pattern of a data region.

   This connects the value that contains your XML contents from your parent report to the parameter in your subreport.

   When you have completed this, XML data will be pulled by the parent report and exposed to the subreport using a parameter. This will then be added to the query, which the XML data provider will reference directly.

Limitations and Common Pitfalls

Provided the previous information and examples, new report authors might still have difficulties when querying XML data sources. Some of the most common misunderstandings include the following:

- **Query syntax**—The XML data-provider query syntax is incompatible with XPATH. For more information, see the XML Data Provider Query Language section, earlier in this white paper.
- **Multiple parent-child or master-detail relationships at the same level**—XML supports this construct. However, when flattening to a two-dimensional data set, the provider cannot support this in one query. For example, if a customer has multiple returns and multiple orders, the provider can return only one set: orders or returns. However, this can be combined in one report by using two data sets. Multiple datasets let you display the data within the same report, but in separate data regions.
- **XML schema of Web service response**—The structure of Web service responses is typically not well-documented; it can be unclear how to construct the query against a Web service. For examples about how to determine the structure, see the Querying a Web Service example, earlier in this white paper.

- **Web service encoding**—Frequently, Web services will use HTML or Base64 encoding in their responses. The XML Data Provider supports both decoding mechanisms through the query syntax. For more information, see the XML Data Provider Query Language section, earlier in this white paper.
- **SOAP Action auto-derivation**—The XML Data Provider will auto-generate SOAP Action by appending Method Name and Namespace. For non-.NET Framework Web services, the SOAP Action may differ and will have to be set explicitly in the query.
- **XML namespaces**—Frequently, consumers of Web services will have to address namespaces in the source XML. Setting **IgnoreNamespaces=true** eases the problem. However, if there are specific requirements that require the use of namespaces, see the XML Data Provider Query Language section, earlier in this white paper.
- **Type casting**—Unless specified in the query, data types will be returned as strings. The data provider will not automatically detect the type information. This problem usually occurs when authors try to use data in aggregation expressions. However, if the types of the fields can be explicitly specified, see the XML Data Provider Query Language section, earlier in this white paper.

Conclusion

Microsoft SQL Server 2005 Reporting Services offers integration with heterogeneous environments through XML and Web services. This white paper has provided general guidance and useful tips for using these sources. For more information and specifics about how to use XML data sources in addition to other Reporting Services features, see SQL Server Books Online.

For more information:

Microsoft SQL Server Developer Center [http://go.microsoft.com/fwlink/?LinkID=42457]