

Operations Manager 2007 R2 Report Authoring Guide

Microsoft Corporation

Published: June 2011

Authors

Daniel Savage, Liza Poggemeyer, Brian MacDonald

Feedback

Send suggestions and comments about this document to [momdocs@microsoft.com](mailto:momdocs@microsoft.com?subject=Report%20%20Authoring%20Guide%20published%20Jul%202011). Please include the Report Authoring Guide name and publish date with your feedback.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, SQL Server, Windows, and Windows Server and are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are property of their respective owners.

Revision History

|  |  |
| --- | --- |
| Release Date | Changes |
| February 26, 2008 | Original release of this guide |
| February 29, 2008 | Corrected the syntax of the query to create a custom report |
| April 11, 2008 | Clarified the procedures available in the scenario to create and publish a report. |
| September 30, 2008 | Added information about publishing reports to a SharePoint site, as well as improved the information about scheduling reports. |
| June 20, 2011 | Removed content on Linked Reports that has been moved to Management Pack Authoring Guide.  Rewrite of sections on creating custom report. Provided greater detail and use of Authoring console. |

Contents

[Operations Manager 2007 Report Authoring Guide 6](#_Toc298844937)

[Overview of Custom Reports 6](#_Toc298844938)

[Setting up the Environment 8](#_Toc298844939)

[Creating Custom Reports 10](#_Toc298844940)

[The OperationsManagerDW Schema 12](#_Toc298844941)

[Inside a Generic Report 21](#_Toc298844942)

[Creating Custom Queries 33](#_Toc298844943)

[Using a Report Builder 41](#_Toc298844944)

[Custom Report Parameters 46](#_Toc298844945)

[Localizing Reports 63](#_Toc298844946)

[Using Stored Procedures in a Custom Report 72](#_Toc298844947)

[Deploying Reports 76](#_Toc298844948)

[Scheduling and Sending Reports 80](#_Toc298844949)

[Creating a Report Schedule 80](#_Toc298844950)

[Sending a Report Through E-mail 81](#_Toc298844951)

[Delivering a report to the SQL Server Report Server Cache 82](#_Toc298844952)

[Appendix A - Reporting Management Pack Schema 83](#_Toc298844953)

[Appendix B - Parameter Values and Parameter Controls 86](#_Toc298844954)

[Appendix C - Data Types and Sample Queries 106](#_Toc298844955)

Operations Manager 2007 Report Authoring Guide

This guide provides the details for creating a custom report in Operations Manager 2007 R2. You should use this guide if you have reporting requirements that cannot be addressed with one of the reports provided in the management packs that you have installed in your management group or by one of the generic reports that are installed by default.

Related Content

The Operations Manager 2007 Reporting Guide provides general information on the reporting feature and describes how to perform functions such as running or scheduling an existing report.

The Management Pack Authoring Guide provides complete documentation on Linked Reports which allow you to include an existing report customized for your application in your management pack.

Overview of Custom Reports

Custom reports are intended for advanced users who are comfortable creating their own reports by using Microsoft SQL Server Reporting Services and SQL Server Business Intelligence Development Studio, SQL Server Reporting Services Report Builder, or Microsoft Visual Studio Report Designer.

If report types using prebuilt queries do not deliver the reporting data that you need, use custom reports by creating the queries yourself to retrieve just the information you want. For example, you can create a query that consolidates the data from multiple objects into a single report. You can then customize the appearance of the report, using one of several tools or programs that include Microsoft SQL Server 2008 Business Intelligence Development Studio, Visual Studio Report Designer, Microsoft SQL Server Reporting Services Report Builder, or Excel, to fit the needs of your organization. In addition, you can localize custom reports for multiple languages. After you have deployed the report, you and other users can run the report like any other report.

Considerations for Choosing the Appropriate Type of Report

Before you create a custom report, you must first decide what information you need in the report. Determine what counters you want to monitor, and what parameters you want to report on. For example, if you are reporting on availability, what time period should the report cover? If you are reporting on performance, what is the acceptable threshold for the counter? Next, determine the appearance of the report. Simple numeric data might be sufficient, or you might require a more advanced graph that uses customized images, such as your corporate logo.

After you have concluded these determinations, collect feedback from users who will use this report. Their input might lead you to revise the report design.

 If your report concerns one type of data, such as an Availability Report, see the existing Generic Reports. All counters and performance data stored in the data warehouse are available through one of the generic reports.

 If your report uses specific parameters that users should not change, for example, availability data over the last week, use a linked report. Both generic and linked reports can be created and used without knowledge of SQL Server or the data warehouse. For more information about linked reports, see Linked Reports.

 If your report uses multiple types of objects, or multiple types of data, such as performance and availability in a single report, or if you need a report that has a customized appearance, use a custom report. Because custom reports require a higher level of familiarity with Transact-SQL and the data warehouse schema, we recommend that you only use a custom report if your report has these requirements and you are an advanced user.

Prerequisites for Custom Reports

To write a custom report, you must install the following services and programs on your report development server:

 System Center Operations Manager 2007 Reporting Services

 SQL Server 2005 Reporting Services or SQL Server 2008 Reporting Services

 SQL Server 2005 Business Intelligence Development Studio or SQL Server 2008 Business Intelligence Development Studio, SQL Server 2005 Reporting Services Report Builder or SQL Server 2008 Reporting Services Report Builder, or Visual Studio 2008 Report Designer

Overview of Custom Report Authoring

To write a custom report, follows these basic steps:

1. Write a Transact-SQL query that retrieves the information that you need.

2. Add parameters to the query for operators to use to run the report.

3. Design the report in SQL Server Report Designer or Visual Studio Report Designer.

4. Optionally, load the report definition and Report Definitions Language (RPDL) file into a management pack for distribution.

5. Import the report or the management pack.

For more information about how to create custom reports, see [Creating Custom Reports](#z4f0322e700ed4480a386edc409a7447a).

In This Section

[Setting up the Environment](#z50474d3c3da644c480600afc8cb7863d)

|  |
| --- |
| Describes the installation requirements for setting up Operations Manager 2007 Reporting Services and explains how to create a read-only user and how to create a new data source. |

[Creating Custom Reports](#z4f0322e700ed4480a386edc409a7447a)

|  |
| --- |
| Provides the background and walkthroughs for creating custom reports and describes how to write a custom query, how to use the Business Intelligence Development Studio, and how to set up the report parameters area. |

[Deploying Reports](#z18c5f1eb721f45209918d45203a3ef7f)

|  |
| --- |
| Describes how to deploy reports to the Operations Manager Reporting Server or to save to custom management packs for deployment to other Operations Manager installations. |

Setting up the Environment

Operations Manager 2007 Reporting requires that you install Microsoft SQL Server 2005 or SQL Server 2008 and SQL Server Reporting Services (SSRS). Because Operations Manager 2007 Reporting includes SQL Server Reporting Services, you do not need to install it separately. For more information about Operations Manager 2007 Reporting, see the [Operations Manager 2007 Deployment Guide](http://go.microsoft.com/fwlink/?LinkId=204116) in the System Center Operations Manager 2007 R2 Technical Library. The default configuration of SQL Server 2005 and SQL Server 2008 includes Business Intelligence Development Studio (BIDS).

BIDS is a subset of Microsoft Visual Studio 2008 designed specifically for use with SQL Server 2008. It has various tools for SQL Server users including the robust Report Designer. You can create custom reports for use in Operations Manager with a variety of tools that include SQL Server Reporting Services Report Builder, Visual Studio, and Microsoft Excel. This guide focuses on BIDS, but you can use the tool that you are most comfortable with. For more information about the tools in BIDS, see [Introducing Business Intelligence Development Studio](http://go.microsoft.com/fwlink/?LinkId=204117) in the SQL Server 2008 R2 Books Online (http://technet.microsoft.com/en-us/library/ms173767.aspx).

If you already have Visual Studio 2008 installed on your Operations Manager Reporting server, Business Intelligence Design Studio uses Visual Studio instead.

A common practice for authoring reports is to create a read-only account in SQL Server that enables multiple authors to access the OperationsManagerDW database to test their reports without giving each author individual access. To create a read-only data source, you first must create a database user that only has Read permissions to the OperationsManagerDW database.

To create a database user that has only Read permission

|  |
| --- |
| 1. Open Microsoft SQL Server Management Studio (SSMS) and connect to the database server.  2. In Object Explorer, right-click the server name, and then click Properties. When the Server Properties dialog box opens, click the Security tab.  3. In the Server authentication section, ensure that SQL Server and Windows Authentication mode is selected. Click OK. You might have to restart SQL Server for the change to take effect. Close the Server Properties dialog box.  4. In Object Explorer, expand Security, right-click Logins, and then click New login. The Login – New dialog box appears. Enter a login name, select SQL Server Authentication, and enter a password. Clear the User must change password at next login check box.  5. Select the User Mapping page on the left. In the Users mapped to this login box, select OperationsManagerDW. The box at the bottom of the dialog box changes to Database role membership for: OperationsManagerDW. Select the db\_datareader role. Click OK. |

As an alternative to creating a database user that has Read-Only permission, especially in environments where operating in mixed mode is not desired, you can create a login for each operator’s Active Directory account, and then grant that login db\_datareader role membership on the OperationsManagerDW database.

To set up BIDS to create reports, you must create a new data source to connect to the reporting data warehouse.

To create a new report project

|  |
| --- |
| 1. On the Start menu, point to SQL Server 2008, and then click SQL Server Business Intelligence Development Studio.  2. On the Start page, click File, click New, and then click Project, or click the Create: Project link. This opens the New Project dialog box.  3. In the Project types pane, ensure that Business Intelligence Projects is selected. In the Templates pane, select Report Server Project. In the Name box, enter a name for the project. Change the location where the project is stored or accept the default. Click OK.  The new report project is now ready for you to create a data source. |

To create a new data source

|  |
| --- |
| 1. After you create a new report project, the Solution Explorer and Properties panes appear on the right side of the Start page. In the Solution Explorer, right-click Shared Data Sources, and then click Add New Data Source.  2. In the Shared Data Source Properties dialog box, enter a Name for the data source. Leave the Type set to Microsoft SQL Server. Click the Edit button next to the Connection String. The Connection Properties dialog box appears.  3. In the Server name box, enter the name of the server where the reporting data warehouse is installed. If you created a user that has Read-Only permissions, in the Log on to the Server section, select the Use SQL Server Authentication button, and enter the credentials for the SQL Server Login that you created. If not, use whatever authentication is appropriate in your environment. In the Connect to a Database section, select the Select or enter a database name button, and then select OperationsManagerDW from the box. If you do not see OperationsManagerDW, verify that you have selected the correct server and authentication.  4. Click OK. The connection string in the Shared Data Source Properties dialog box will be filled in for you. Click OK to close this dialog box. The data source now appears in the Properties pane, with the extension .rds.  5. To make the data source available to other projects, right-click the data source in the Solution Explorer and click Properties. The Property Pages window for this data source opens. Click TargetServerURL and enter the URL for the report server where the project is deployed. Click OK.  6. Right-click the data source in the Solution Explorer and click Deploy. The Output window shows the progress of the deployment.  The data source is now ready to use with a custom report. |

Creating Custom Reports

If the generic reports provided with Operations Manager do not meet your needs, you can create a custom report. For example, you may need to report on multiple types of objects or multiple types of data, such as performance and availability, in a single report; or, you may need a report that has a customized appearance. The following sections will show you how to create a SQL query that retrieves the information you want, design the report in a report design tool, and set up parameters so operators can customize the report.

Operations Manager Databases

Operations Manager uses two separate databases:

OperationsManager database

|  |
| --- |
| This database stores alerts, discovered inventory, performance data, state data, events, and metadata that is collected by the agents and passed to the Operations console. It also contains information about the management group and management packs. This is information that the operator needs to see and respond to in real time. Any data viewed in the Operator’s Console outside the reporting workspace is stored in this database. |

OperationsManagerDW database

|  |
| --- |
| This database stores data for use in reporting. By default, performance data is sent to this database, as are alerts, events, and state information. This database is updated regularly with the latest information from the OperationsManager database to ensure that the data is current. |

The OperationsManagerDW database, also called the Data Warehouse, is the focus of this section. The OperationsManagerDW database is optimized for reporting. It stores raw data from the monitors and rules for a limited time only. By default, to improve performance, reports only access aggregated data. There are daily and hourly aggregations. See Kevin Holman’s blog for more information about [Data Warehouse grooming](http://go.microsoft.com/fwlink/?LinkId=204249): http://go.microsoft.com/fwlink/?LinkId=204249.

The OperationsManagerDW Schema

The most important part of writing a custom report is generating the correct query to extract the information you need from the OperationsManagerDW database. To write the correct query, you need to understand the database schema. Reports should only query the database views built into OperationsManagerDW; you should not access the tables directly.

In This Section

[The OperationsManagerDW Schema](#z4875aec171704baa8941adf574ba5444)

|  |
| --- |
| Before you can start to write a custom report, you need to know what views in the OperationsManagerDW database hold the information you want to retrieve. For custom queries, you only need to access a subset of the schema. This section covers this subset of the schema and introduces the views that you’ll need to access in order to write a custom query. |

[Inside a Generic Report](#z370c08973d5c437f9138ffa00eabd86a)

|  |
| --- |
| A good way to become familiar with the SQL queries you’ll need in order to create custom reports is to look inside one of the published reports that are installed with the Operations Manager Management Pack. This section examines the SQL queries in several of the reports and explains how they function, which will enable you to build your own queries. |

[Creating Custom Queries](#zc41f0cb483d04c5085fa2bfd935c23e3)

|  |
| --- |
| Once you understand the schema and how the queries work in the existing reports, this section shows you how to write your own custom report query. This section will use multiple examples, starting with a basic query that retrieves just one counter across several computers, and then moving to more complex examples. Although these examples address specific custom reporting needs, the intent is that you will be able to use them as a model to create your own custom queries. |

[Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b)

|  |
| --- |
| Operations Manager reports contain a parameter area that allows users to specify the bounds of the database query with respect to target objects, monitoring times, and other options. You can define what parameters are available in your custom report, and populate those parameters with the default data that users are most likely to want. This section explains how to customize the parameter area of your custom report. |

[Localizing Reports](#z52a192a059d3425397432a1eebb88ea4)

|  |
| --- |
| This section describes how to localize management packs for multiple language support. |

[Using Stored Procedures in a Custom Report](#zecb3841d1f1f43beb92685ab92e8578b)

|  |
| --- |
| Stored procedures provide more convenience and security than using embedded Transact-SQL queries in a report, but they also require some additional configuration. This section describes how to create stored procedures, use them in reports, and create scripts to enable stored procedures to be distributed with management packs. |

[Using a Report Builder](#zbbd29df9345b4b0697f6edc07f80ab47)

|  |
| --- |
| Unlike linked reports, custom reports require that you design the report using a report designer, either in SQL Server Business Intelligence Development Studio (BIDS), Visual Studio Report Developer, or SQL Server Report Builder. This section explains how to use BIDS to create clear and useful reports. |

The OperationsManagerDW Schema

The OperationsManagerDW database stores data for use in reports. This is in contrast to the OperationsManager database, which stores data for use at the Operations Manager console. A large number of database views are provided in order to make queries easier to write, without requiring you to have detailed knowledge of the schema.

Datasets

There are five main groupings, or datasets, within the OperationsManagerDW database:

Alert

|  |
| --- |
| Contains records of alerts raised by the management pack rules and monitors. |

Event

|  |
| --- |
| Contains records of event data collected by management pack rules. |

Performance

|  |
| --- |
| Contains data regarding the performance of managed entities (objects monitored by Operations Manager), collected by management pack rules. |

State

|  |
| --- |
| Contains data collected by monitors about the state of a managed entity. |

ManagedEntity

|  |
| --- |
| Provides information about monitored objects, management packs, management groups, and relationships. The views in this dataset are typically joined to other views (Alert, Event, Performance, and so on) to provide the name of the monitored object, or to scope the query to a specific group, management pack, or management group. |

Each of these datasets corresponds to a different category of reports in the operator console.

Database Views

The tables of the OperationsManagerDW database should never be accessed directly. Instead, a number of database views are provided for access. When structural changes are made to the OperationsManagerDW database, every effort is made to ensure that the views are consistent so that existing reports can be used without changes. The tables, however, may be changed, and these changes may cause existing reports that query the tables directly to no longer work properly. For a full list of the views that should be used, see the [AuthorMPs web site](http://go.microsoft.com/fwlink/?LinkId=204120) (http://go.microsoft.com/fwlink/?LinkId=204120).

Key Views for Each Dataset

Alert Dataset

Alerts are generated by rules and monitors in Operations Manager and are stored in the database together with their respective details and parameters. No aggregation or summarization is performed. Alert data can be seen in four views in the Data Warehouse database:

Alert.vAlert

|  |
| --- |
| This view contains general information about the alert: Name, Description, Severity, the workflow that created it, the time that it was created, and the managed entity that it is associated with. This view can be joined to the other alert views on the AlertGuid column. It may also be joined to the ManagedEntity, Rule, and Monitor views to retrieve additional details. |

Alert.vAlertDetail

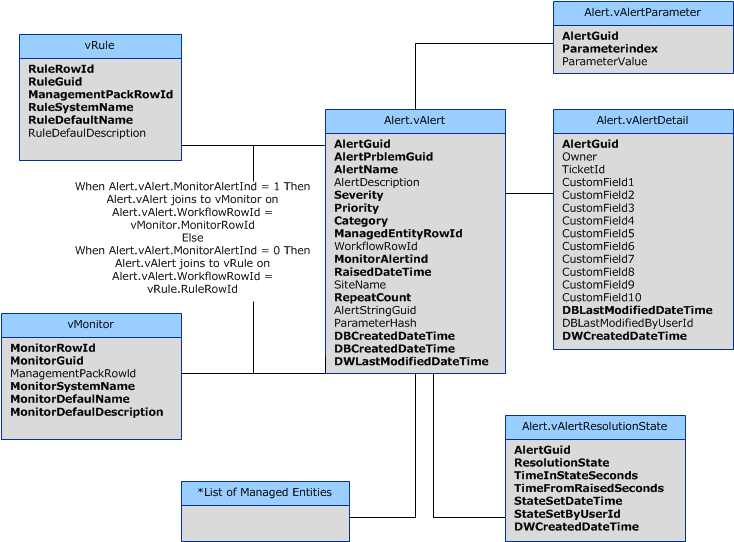
|  |
| --- |
| This view contains the Custom Field, Owner, and TicketID data for the alert. It can be joined to the Alert.vAlert view on the AlertGuid column. |

Alert.vAlertResolutionState

|  |
| --- |
| This view contains information about each Resolution State that the alert has been in, when it was set, and how long it was in the state. This view can be joined to the Alert.vAlert view on the AlertGuid column. |

Alert.vAlertParameter

|  |
| --- |
| This view contains the value for each parameter in the alert. It can be joined to the Alert.vAlert view on the AlertGuid column. |



Sample Alert Dataset Query

To retrieve the number of alerts generated yesterday, use the following Transact-SQL query:

SELECT COUNT (AlertName)

FROM

Alert.vAlert

WHERE DATEDIFF(dd,0,RaisedDateTime) = DATEDIFF(dd,0,GETDATE()-1)

Event Dataset

Events are collected from monitored objects by Event Collection Rules in Operations Manager and are stored in the database together with their respective details and parameters. No aggregation or summarization is performed. Event data can be seen in several views in the Data Warehouse database:

Event.vEvent

|  |
| --- |
| This is the main view for Events. It contains the unique ID for the Event (EventOriginId), the date/time of the Event, the Event Number, and other unique IDs that can be joined to the other Event views to get the details of the Event. |

Event.vEventDetail

|  |
| --- |
| This view can be joined to the Event.vEvent view on the EventOriginId column to provide the event description found in the EventData column. |

Event.vEventParameter

|  |
| --- |
| This view can be joined to the Event.vEvent view on the EventOriginId column to provide the event parameters. This is generally the same information that is in the event description, with part of it in each parameter. Separate joins should be done from the Event.vEvent view for each parameter that is needed. |

Event.vEventRule

|  |
| --- |
| This view contains the RuleRowId of the rule that generated the event and the ManagedEntityRowId of the managed entity that the alert came from. This view should be joined to the Event.vEvent, vRule, and vManagedEntity views to get these details. |

vEventCategory

|  |
| --- |
| This view provides the event category information and can be joined on the EventCategoryRowId column in the Event.vEvent view. |

vEventLevel

|  |
| --- |
| This view provides the event Level (Warning, Error, and so on) and can be joined on the EventLevelId column in the Event.vEvent view. |

vEventUserName

|  |
| --- |
| This view provides the user name that was logged with the event and can be joined on the UserNameRowId column in the Event.vEvent view. |

vEventChannel

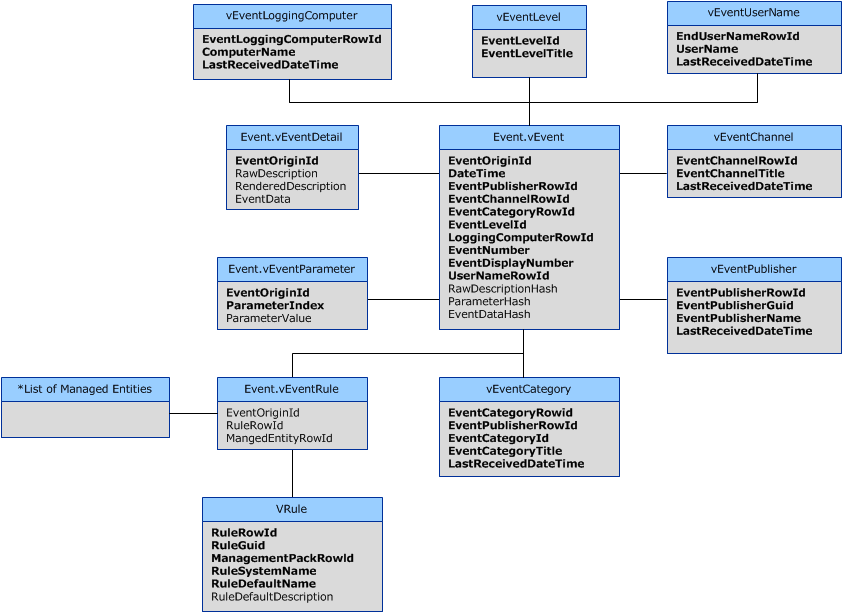
|  |
| --- |
| This view provides the channel for the event. This will be either the name of the event log (Application, System, and so on) or a name supplied in a custom event collection rule. This view can be joined on the EventChannelRowId column in the Event.vEvent view. |

vEventPublisher

|  |
| --- |
| This view provides the publisher for the event. This will be either the Source in the event log or a name supplied in a custom event collection rule. This view can be joined on the EventPublisherRowId column in the Event.vEvent view. |

vEventLoggingComputer

|  |
| --- |
| This view provides the name of the computer that logged the event and can be joined on the LoggingComputerRowId column in the Event.vEvent view. |



Sample Event Dataset Query

To retrieve the ten most frequently occurring events in the OperationsManagerDW database, use the following Transact-SQL query:

SELECT TOP 10 EventDisplayNumber, COUNT(\*) AS Events

FROM Event.vEvent

GROUP BY EventDisplayNumber

ORDER BY Events DESC

Performance Dataset

Performance data is available in daily and hourly aggregates, which contain the minimum, maximum, average, and standard deviation of the values collected. You will generally query either the Perf.vPerfDaily or Perf.vPerfHourly views and join them to other views to get the managed entity and rule information. The vPerfRaw view can be queried (instead of the hourly or daily views) to get the value of each sample. By default, this data is kept for a shorter time than the hourly and daily aggregated data. For optimal report performance, it is recommended that you use vPerfDaily or vPerfHourly instead of vPerfRaw. The Performance data views are as follows:

Perf.vPerfDaily and Perf.vPerfHourly

|  |
| --- |
| These views contain the hourly and daily aggregated performance data. The minimum, maximum, average, and standard deviation of the values in the vPerfRaw view are calculated and stored in these views. These views can be joined to the vManagedEntity table on the ManagedEntityRowId column to get the information for the object that the values were collected from. They can then be joined to the vPerformanceRuleInstance view on the PerformanceRuleInstanceRowId column to get the name of the performance object and counter and the rule that was used to collect them. |

vPerformanceRuleInstance

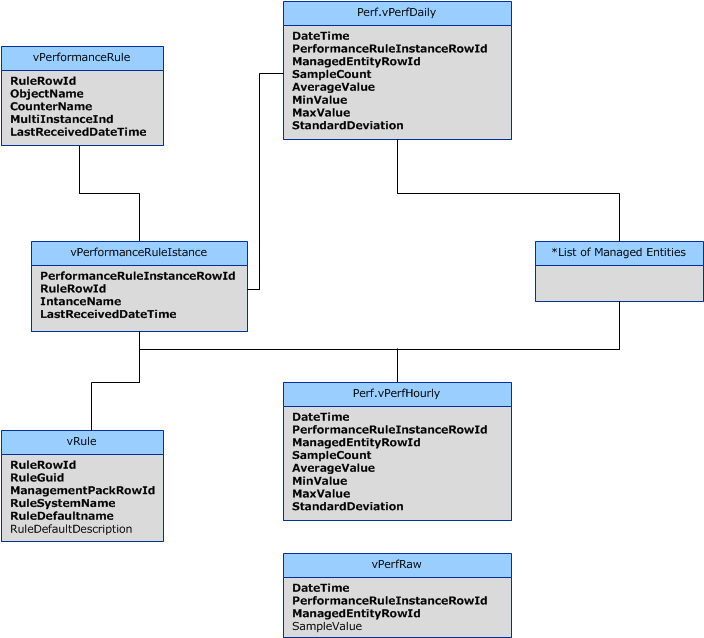
|  |
| --- |
| This view contains the ID for the instance of the counter that was collected together with the IDs for the performance counters and the rules that collected them. Use this view to link the daily/hourly aggregation views to the vPerformanceRule view on the PerformanceRuleInstanceId column, and then to the vRule view on the RuleRowId column. |

vPerformanceRule

|  |
| --- |
| This view contains the counter and object names that were collected and can be joined to the PerformanceRuleInstanceRowId column in the vPerformanceRuleInstance view. |

vRule

|  |
| --- |
| This view contains the name of the rule that collected the performance counter and can be joined on the RuleRowId column in the vPerformanceRuleInstance view. It also contains a ManagmentPackRowId column which can be joined to the vManagementPack view to get the information about the management pack that contains the rule. |



Sample Performance Dataset Query

To retrieve the top ten performance rule instances being monitored, use the following query:

SELECT TOP 10 PerformanceRuleInstanceRowId, COUNT(\*) AS Count

FROM Perf.vPerfDaily

GROUP BY PerformanceRuleInstanceRowId

ORDER BY Count DESC

State Dataset

The State data views contain data on how long any given monitor was in each possible state (Healthy, Warning, Critical, and so on). The data is summarized for each object and aggregated daily and hourly. The State data views are described and illustrated below:

vStateHourlyFull and vSTateDailyFull

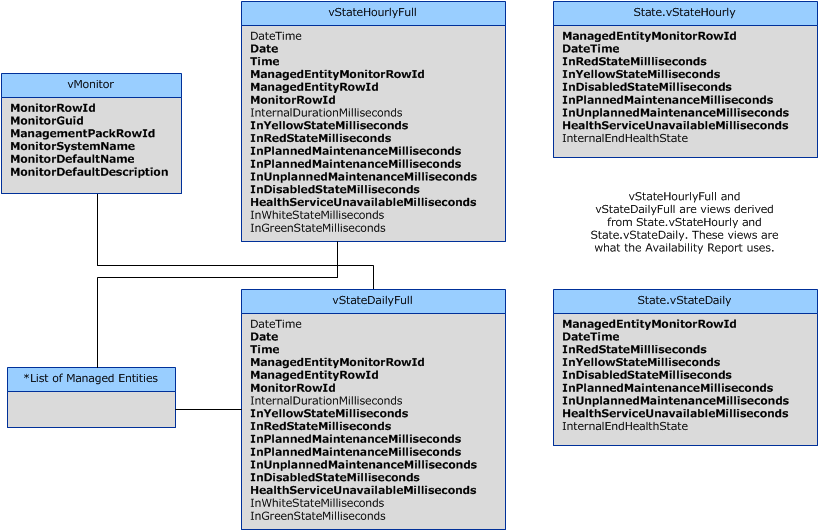
|  |
| --- |
| These views contain the hourly and daily aggregated state data and contain data on how long each monitor/managed entity combination was in each possible state. The MonitorRowId and ManagedEntity columns can be joined to the vMonitor and vManagedEntity views to get information about the monitor that changed the state and the managed entity that the state was changed for. It is not recommended that you use these views in Availability reports; use State.vStateHourly and State.vStateDaily instead. |

State.vStateHourly and State.vStateDaily

|  |
| --- |
| These views are derived from the vStateHourlyFull and vStateDailyFull and are used in availability reports. These views can be joined to the vManagedEntityMonitor on the ManagedEntityMonitorRowId column. |

vMonitor

|  |
| --- |
| This view contains the information about the monitor that caused the state change and can be joined on the MonitorRowId column of the vStateHourlyFull and vStateHourlyDaily views. |



Sample State Dataset Query

To retrieve the ManagedEntityMonitorRowID for any entity that has been in the red (Critical) state for more than one minute, use the following query:

SELECT ManagedEntityMonitorRowId

FROM State.vStateHourly

WHERE InRedStateMilliseconds > 60000

Managed Entity Dataset

The managed entity dataset provides information about monitored objects, management packs, relationships, and management groups. These views are typically joined to other views (Alert, Event, Performance, or State) to provide the name of the monitored object or to scope the query to a specific group, management pack, or management group.

vManagedEntity

|  |
| --- |
| This view provides the names for all monitored objects in the OperationsManagerDW database. This view is generally joined to views from the other datasets to provide the name of the object that the collected data applies to. This view can also be joined to the vManagedEntityType view on the ManagedEntityTypeRowId column to scope the list of objects down to a specific type of object, to the vRelationship view on the ManagementGroupRowId column to scope the list of objects down to a specific group, and to the vManagementGroup view on the ManagementGroupRowId column to scope the list of objects to a specific management group. |

vManagedEntityType

|  |
| --- |
| This view provides information about the types of objects that occur in the vManagedEntity view (operating system, database, and so on). This view can be joined to the vManagedEntity view on the ManagedEntityTypeRowID column. This view can also be joined to the vManagementPack view on the ManagementPackRowId column to show which management pack defines each type of object. |

vManagementPack

|  |
| --- |
| This view provides information about the Management Packs in the Operations Manager environment and can be joined to the vManagedEntityType, vMonitor, and vRule views. |

vRelationship

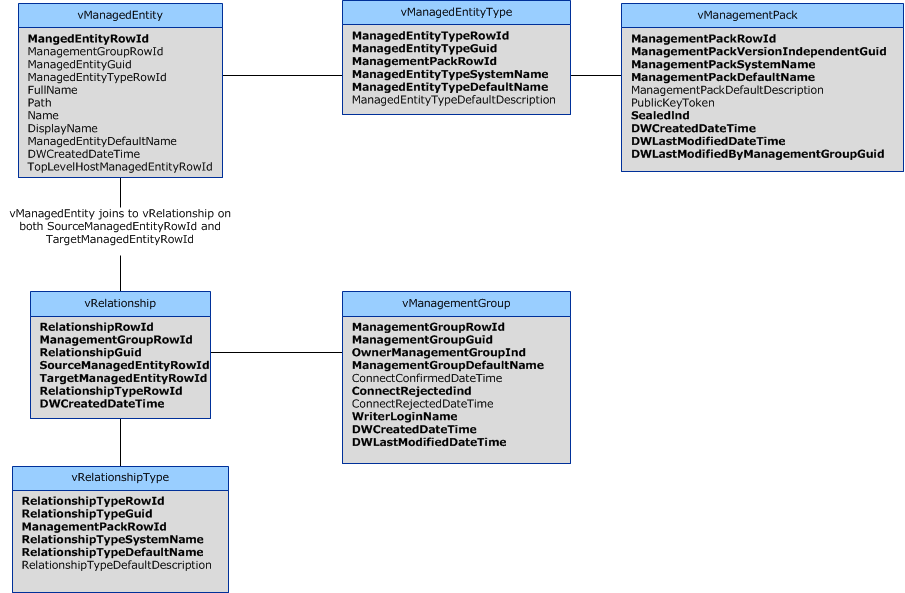
|  |
| --- |
| This view provides relationship information, which can be used to retrieve group membership for Managed Entities. This view can be joined to the vManagedEntity view on the SourceManagedEntityRowId and TargetManagedEntityRowId columns. |

vRelationshipType

|  |
| --- |
| This view provides information about the type of relationships identified in the vRelationship view and can be joined to other views on the RelationshipTypeRowId column. |

vManagementGroup

|  |
| --- |
| This view provides information about Management Groups that send data to the Data Warehouse and can be joined to several other views on the ManagementGroupRowId column. |



Inside a Generic Report

Before you start building your own queries, it may be helpful to see how the queries for the built-in reports work. The sections below look at the queries used by three existing reports from the Microsoft Generic Report Library: The Availability report, the Alerts report, and the Performance report. For each report, the query is located in a stored procedure. The stored procedures are reproduced here so you can see the Transact-SQL code.

The Alert Report

The purpose of the Alert report is to present a simple list of alerts raised by the managed objects specified in the parameters of the report. The report in the Generic Library can list events of all severities (Information, Warning, and Critical) and all Priorities (Low, Medium, and High). For each alert, it provides the Alert Name, Repeat Count, Priority, the Object that raised the alert, and the first and last dates where the alert appeared.

This query deals with Alert data, so it uses the vAlert view. First, it retrieves the AlertName, Severity, and Priority, which are all basic information. It also retrieves the ManagedEntityRowId, which will be used as a foreign key later in the query:

SELECT

Alert.vAlert.AlertName,

Alert.vAlert.Severity,

Alert.vAlert.Priority,

Alert.vAlert.ManagedEntityRowId,

Each time the alert is raised it creates a new record. To get the first and last dates where the alert was raised, this query uses the MAX and MIN functions:

MAX(Alert.vAlert.RaisedDateTime) AS LastRaisedTime,

MIN(Alert.vAlert.RaisedDateTime) AS FirstRaisedTime,

The query also uses the MIN function to retrieve the AlertDescription. The description is just text, but using MIN guarantees that the query will retrieve only one entry for the given alert. This is required because the descriptions should all be identical:

MIN(Alert.vAlert.AlertDescription) AS AlertDescription,

To get the number of times an alert was repeated, the query uses the COUNT function. This simply counts the number of rows retrieved and indicates how many times the event was raised:

COUNT(\*) AS RepeatCount,

Finally, the query retrieves the SiteName and the AlertProblemGuid. It does not use the AlertGuid because that row is unique for each individual Alert. Instead, it uses the AlertProblemGuid, which identifies the specific type of alert:

Alert.vAlert.SiteName,

Alert.vAlert.AlertProblemGuid

To define the FROM clause, the query retrieves all of the rows above from the vAlert view. It then joins that information to several temporary tables. The #ObjectList table holds the objects passed in to the query from the report. The vAlert.Severity and vAlert.Priority fields are simply integers, which can be used with the #SeverityList.Severity field and the #PriorityList.Priority tables to return text strings:

FROM Alert.vAlert

INNER JOIN #ObjectList ON #ObjectList.ManagedEntityRowId = Alert.vAlert.ManagedEntityRowId

INNER JOIN #SeverityList ON #SeverityList.Severity = Alert.vAlert.Severity

INNER JOIN #PriorityList ON #PriorityList.Priority = Alert.vAlert.Priority

The WHERE clause simply limits the query by the dates and the site name (if any) that was passed from the report form. The variables preceded by the @ symbol are parameters defined in the report form.

WHERE

Alert.vAlert.RaisedDateTime >= @StartDate AND

Alert.vAlert.RaisedDateTime < @EndDate AND

(@SiteName IS NULL OR Alert.vAlert.SiteName = @SiteName)

Finally, there’s a simple GROUP BY clause which groups records with identical results into summary rows, so that the report does not contain duplicate alerts:

GROUP BY

Alert.vAlert.AlertName,

Alert.vAlert.Severity,

Alert.vAlert.Priority,

Alert.vAlert.ManagedEntityRowId,

Alert.vAlert.SiteName,

Alert.vAlert.AlertProblemGuid

Here’s what the complete query looks like:

SELECT

Alert.vAlert.AlertName,

Alert.vAlert.Severity,

Alert.vAlert.Priority,

Alert.vAlert.ManagedEntityRowId,

MAX(Alert.vAlert.RaisedDateTime) AS LastRaisedTime,

MIN(Alert.vAlert.RaisedDateTime) AS FirstRaisedTime,

MIN(Alert.vAlert.AlertDescription) AS AlertDescription,

COUNT(\*) AS RepeatCount,

Alert.vAlert.SiteName,

Alert.vAlert.AlertProblemGuid

FROM Alert.vAlert

INNER JOIN #ObjectList ON #ObjectList.ManagedEntityRowId = Alert.vAlert.ManagedEntityRowId

INNER JOIN #SeverityList ON #SeverityList.Severity = Alert.vAlert.Severity

INNER JOIN #PriorityList ON #PriorityList.Priority = Alert.vAlert.Priority

WHERE

Alert.vAlert.RaisedDateTime >= @StartDate AND

Alert.vAlert.RaisedDateTime < @EndDate AND

(@SiteName IS NULL OR Alert.vAlert.SiteName = @SiteName)

GROUP BY

Alert.vAlert.AlertName,

Alert.vAlert.Severity,

Alert.vAlert.Priority,

Alert.vAlert.ManagedEntityRowId,

Alert.vAlert.SiteName,

Alert.vAlert.AlertProblemGuid

Although this query may seem complicated, its function isn’t. It retrieves most of the values from the vAlert view, filtered by date, with additional access to some outside lists.

The Availability Report

In contrast to the Alert report, the Availability report draws information from several different views. The Availability report shows the time in any given state (Red, Yellow, Green, White, Disabled, Planned Maintenance, and Unplanned Maintenance) for the monitored entities.

This report returns availability data, which is a state condition, so the information is in the vState views. The query starts off by selecting the amount of time in each state:

SELECTvState.InRedStateMilliseconds,

vState.InYellowStateMilliseconds,

vState.InGreenStateMilliseconds,

vState.InWhiteStateMilliseconds,

vState.InDisabledStateMilliseconds,

vState.InPlannedMaintenanceMilliseconds,

vState.InUnplannedMaintentanceMilliseconds,

vState.HealthServiceUnavailableMilliseconds

The report also needs to show which entities the information is associated with. Therefore, the query retrieves the name, ID, and path for the managed entities in the report. The ManagedEntityRowId is again important because it’s used as a foreign key later.

vManagedEntity.ManagedEntityRowId,

vManagedEntity.ManagedEntityDefaultName,

vManagedEntity.ManagedEntityGuid,

vManagedEntity.Path,

The query also retrieves the type of the managed entity by retrieving the GUID for that entity:

vManagedEntityType.ManagedEntityTypeGuid,

The GUID for the managed entity type is enough to identify it but the report also displays the name for that type. If the management pack author did not include a display name, this field will not be available so the stored procedure uses an ISNULL statement to retrieve the name if it is available. If not, it retrieves the default name instead:

ISNULL(vDisplayString.Name,vManagedEntityType.ManagedEntityTypeDefaultName) AS DisplayName

The Availability report also displays an icon for each managed entity in the table so the query retrieves that image also:

vManagedEntityTypeImage.Image

In addition to the managed entity, the query also retrieves the monitor data:

vManagedEntityMonitor.ManagedEntityMonitorRowId,

The query also retrieves the display name for the monitor, just as it did for the managed entity. If the name isn’t available, the ISNULL statement substitutes the default monitor name.

ISNULL(vDisplayStringMonitor.Name,vMonitor.MonitorDefaultName) AS MonitorDisplayName,

The following two items are related to entity monitor dependencies:

vManagedEntityMonitorDependency.ParentManagedEntityRowId,

vManagedEntityMonitorDependency.[Level],

If the entities being monitoring are selected by Management Group, the query retrieves the ID and name of the group also:

vManagementGroup.ManagementGroupGuid,

vManagementGroup.ManagementGroupDefaultName

The SELECT clause retrieves all the data for display in the report. The next step is to determine where to select the data from. This information is with state data, which means you have a choice between vStateDailyFull and vStateHourlyFull. The Microsoft Generic Report Library uses vStateDailyFull:

FROM vStateDailyFull as vState

The query joins the vManagedEntityMonitor view to the state view in order to retrieve the monitors for the managed entities. The ManagedEntityRowId is the foreign key used:

INNER JOIN vManagedEntityMonitor ON

vState.ManagedEntityMonitorRowId = vManagedEntityMonitor.ManagedEntityRowId

The query also joins the vManagedEntity view. This connects the specified monitors to the managed entities, again using ManagedEntityRowId as the foreign key:

INNER JOIN vManagedEntity ON

vManagedEntityMonitor.ManagedEntityRowID = vManagedEntity.ManagedEntityRowId

The query then joins the vManagedEntityType view using ManagedEntityTypeRowId as the foreign key:

INNER JOIN vManagedEntityType ON

vManagedEntity.ManagedEntityTypeRowId = vManagedEntityType.ManagedEntityTypeRowId

Next, the query joins the vMonitor view to the state data. MonitorRowId is the key:

INNER JOIN vMonitor ON vState.MonitorRowId = vMonitor.MonitorRowId

The query also joins the dependencies:

INNER JOIN #ManagedEntityMonitorDependency AS vManagedEntityMonitorDependency ON

vState.ManagedEntityMonitorRowId = vManagedEntityMonitorDependency.ChildManagedEntityMonitorRowId

Finally, the query joins the management groups to their managed entities using the ManagementGroupRowId as the foreign key:

INNER JOIN vManagementGroup ON

vManagedEntity.ManagementGroupRowId = vManagementGroup.ManagementGroupRowId

The query then retrieves the image to use for the entity type:

LEFT OUTER JOIN vManagedEntityTypeImage

ON vManagedEntity.ManagedEntityTypeRowId = vManagedEntityTypeImage.ManagedEntityTypeRowId

AND vManagedEntityTypeImage.ImageCategory = N'u16x16Icon'

The last few joins retrieve the display strings for the managed entity and the monitor:

LEFT OUTER JOIN vDisplayString ON

vManagedEntityType.ManagedEntityTypeGuid = vDisplaySTring.ElementGuid AND

vDisplayString.LanguageCode = @LanguageCode

LEFT OUTER JOIN vDisplayString vDisplayStringMonitor ON

vMonitor.MonitorGuid = vDisplayStringMonitor.ElementGuid AND

vDisplayStringMonitor.LanguageCode = @LanguageCode

The WHERE clause is the next part of the query:

WHERE (vState.DateTime >= DATEADD(hh, DATEPART(hh, @StartDate) + 1, convert (varchar(8), @StartDate, 112))) AND

(vState.DateTime < DATEADD (hh, DATEPART(hh, @EndDate) +1, convert(varchar(8) @EndDate, 112))) AND

(vState.Date BETWEEN DATEADD(day, -1, @StartDate) AND DATEADD(day, 1, @EndDate))

This clause retrieves all data that falls between the two dates that the report user specified in the date picker. @StartDate and @EndDate are the parameters from the date picker.

1. DATEPART(hh, @StartDate) returns the hour part of the @StartDate variable. This value is increased by 1 to return the next full hour after the time specified in @StartDate.

2. convert(varchar(8), @StartDate, 112) converts the @StartDate parameter to character data (varchar(8)) in the format yyyymmdd; the 112 specifies that format.

3. DATEADD(hh, DATEPART(hh, @StartDate) + 1, convert (varchar(8), @StartDate, 112)) adds the specified hour to the specified date, in yyyymmdd format, rounded up to the next hour.

The effect of the entire WHERE clause is to specify the vState data collected between the specified hours. This WHERE clause can be useful in many different reports, and is used throughout the Microsoft Generic Report Library.

Unlike the Alert Report, the Availability report uses information from several different views, most of which can be joined on the ManagedEntityRowId. Also note how the vManagedEntityType and vMonitor views are joined if you want to include that information in your custom report.

Here’s what the complete query looks like:

SELECT

vState.DateTime,

vState.InRedStateMilliseconds,

vState.InYellowStateMilliseconds,

vState.InGreenStateMilliseconds,

vState.InWhiteStateMilliseconds,

vState.InDisabledStateMilliseconds,

vState.InPlannedMaintenanceMilliseconds,

vState.InUnplannedMaintentanceMilliseconds,

vState.HealthServiceUnavailableMilliseconds

vState.IntervalDurationMilliseconds,

vManagedEntity.ManagedEntityRowId,

vManagedEntityMonitor.ManagedEntityMonitorRowId,

vManagedEntityTypeImage.Image,

vManagedEntity.ManagedEntityDefaultName,

vManagedEntityType.ManagedEntityTypeGuid,

vManagedEntityMonitorDependency.ParentManagedEntityRowId,

vManagedEntityMonitorDependency.[Level],

ISNULL(vDisplayStringMonitor.Name,vMonitor.MonitorDefaultName) AS MonitorDisplayName,

vManagedEntity.ManagedEntityGuid,

vManagedEntity.Path,

ISNULL(vDisplayString.Name,vManagedEntityType.ManagedEntityTypeDefaultName) AS DisplayName

vManagementGroup.ManagementGroupGuid,

vManagementGroup.ManagementGroupDefaultName

FROM vStateDailyFull as vState

INNER JOIN vManagedEntityMonitor ON

vState.ManagedEntityMonitorRowId = vManagedEntityMonitor.ManagedEntityRowId

INNER JOIN vManagedEntity ON

vManagedEntityMonitor.ManagedEntityRowID = vManagedEntity.ManagedEntityRowId

INNER JOIN vManagedEntityType ON

vManagedEntity.ManagedEntityTypeRowId = vManagedEntityType.ManagedEntityTypeRowId

INNER JOIN vMonitor ON

vState.MonitorRowId = vMonitor.MonitorRowId

INNER JOIN #ManagedEntityMonitorDependency AS vManagedEntityMonitorDependency ON

vState.ManagedEntityMonitorRowId = vManagedEntityMonitorDependency.ChildManagedEntityMonitorRowId

INNER JOIN vManagementGroup ON

vManagedEntity.ManagementGroupRowId = vManagementGroup.ManagementGroupRowId

LEFT OUTER JOIN vManagedEntityTypeImage ON

vManagedEntity.ManagedEntityTypeRowId = vManagedEntityTypeImage.ManagedEntityTypeRowId AND

vManagedEntityTypeImage.ImageCategory = N’u16x16Icon’

LEFT OUTER JOIN vDisplayString ON

vManagedEntityType.ManagedEntityTypeGuid = vDisplaySTring.ElementGuid AND

vDisplayString.LanguageCode = @LanguageCode

LEFT OUTER JOIN vDisplayString vDisplayStringMonitor ON

vMonitor.MonitorGuid = vDisplayStringMonitor.ElementGuid AND

vDisplayStringMonitor.LanguageCode = @LanguageCode

WHERE (vState.DateTime >= DATEADD(hh, DATEPART(hh, @StartDate) + 1, convert (varchar(8), @StartDate, 112))) AND

(vState.DateTime < DATEADD (hh, DATEPART(hh, @EndDate) +1, convert(varchar(8) @EndDate, 112))) AND

(vState.Date BETWEEN DATEADD(day, -1, @StartDate) AND DATEADD(day, 1, @EndDate))

The Performance Report

The Performance report is similar to the Availability report in that it retrieves data for a specified time period on specific entities and retrieves the information from various views. The purpose of the Performance report is to report on the change of a value over time.

The query retrieves some basic information from the vPerfDaily view, which, in this query, has the alias of vPerf:

SELECT

vPerf.DateTime,

vPerf.SampleCount,

vPerf.AverageValue,

vPerf.MinValue,

vPerf.MaxValue,

vPerf.StandardDeviation,

The query also retrieves the performance rules located in the vPerformanceRuleInstance view:

vPerformanceRuleInstance.RuleRowId,

vPerformanceRuleInstance.InstanceName,

As in the other queries, data is retrieved from the vManagedEntity view, in particular the ManagedEntityRowId, although the GUID, default name, and path are also retrieved:

vManagedEntity.ManagedEntityRowId,

vManagedEntity.ManagedEntityGuid,

vManagedEntity.ManagedEntityDefaultName,

vManagedEntity.Path,

As in the Availability query, the query retrieves a display name if it exists:

ISNULL(vDisplayString.Name,vManagedEntityType.ManagedEntityTypeDefaultName)

AS DisplayName,

To display management group information, the query accesses the vManagementGroup view:

vManagementGroup.ManagementGroupGuid,

vManagementGroup.ManagementGroupDefaultName,

The query also accesses the vRule and vPerformanceRule views:

vRule.RuleGuid,

ISNULL(vDisplayStringRule.Name,vRule.RuleDefaultName) AS RuleDisplayName,

vPerformanceRule.MultiInstanceInd,

The query also retrieves group information from the temporary table #OptionList:

OptionList.[Group],

OptionList.GroupTitle,

OptionList.Position,

OptionList.ChartScale,

OptionList.ChartType,

OptionList.ChartColor,

OptionList.OptionXml,

vManagedEntityTypeImage.Image

The FROM clause in the Performance query is more complex than in the Alert or Availability queries. It begins with the information retrieved from vPerfDaily:

FROM Perf.vPerfDaily as vPerf

The query joins that to the performance rule view, using PerformanceRuleInstanceRowId as the foreign key:

INNER JOIN vPerformanceRuleInstance

ON vPerformanceRuleInstance.PerformanceRuleInstanceRowId = vPerf.PerformanceRuleInstanceRowId

The query then joins that to the vManagedEntity view, again using ManagedEntityRowId as the key:

INNER JOIN vManagedEntity

ON vPerf.ManagedEntityRowId = vManagedEntity.ManagedEntityRow

Once joined to the vManagedEntity view, the query joins to the vManagedEntityType view to retrieve the type for the managed entity:

INNER JOIN vManagedEntityType

ON vManagedEntityType.ManagedEntityTypeRowId = vManagedEntity.ManagedEntityTypeRowId

The query also joins to the vManagementGroup view using the ManagementGroupRowId as the key:

INNER JOIN vManagementGroup

ON vManagementGroup.ManagementGroupRowId = vManagedEntity.ManagementGroupRowId

To retrieve performance information, the query needs to join the vRule and vPerformanceRule views to the vPerformanceRuleInstance view using the RuleRowId as the key in each case.

INNER JOIN vRule

ON vPerformanceRuleInstance.RuleRowId = vRule.RuleRowId

INNER JOIN vPerformanceRule

ON vPerformanceRuleInstance.RuleRowId = vPerformanceRule.RuleRowId

The query needs to join to the #InstanceList:

INNER JOIN #InstanceList AS InstanceList

ON (vPerformanceRuleInstance.PerformanceRuleInstanceRowId = InstanceList.PerformanceRuleInstanceRowId)

The query needs to join to the #OptionList:

INNER JOIN #OptionList As OptionList

ON (InstanceList.Position = OptionList.Position)

Finally, the query joins the ObjectList to the InstanceList and to the vManagedEntity view using the ManagedEntityRowId as the key:

INNER JOIN #ObjectList AS ObjectList

ON (InstanceList.Position = ObjectList.Position)

AND (vManagedEntity.ManagedEntityRowId = ObjectList.ManagedEntityRowId)

The query then retrieves the image to use for the entity type:

LEFT OUTER JOIN vManagedEntityTypeImage

ON vManagedEntity.ManagedEntityTypeRowId = vManagedEntityTypeImage.ManagedEntityTypeRowId

AND vManagedEntityTypeImage.ImageCategory = N'u16x16Icon'

Finally, the query joins the vDisplayString and vDisplayStringRule views to retrieve the necessary display strings for the report. The @LanguageCode is a parameter specified by the report designer.

LEFT OUTER JOIN vDisplayString

ON vManagedEntityType.ManagedEntityTypeGuid = vDisplayString.ElementGuid

AND vDisplayString.LanguageCode = @LanguageCode

LEFT OUTER JOIN vDisplayString vDisplayStringRule

ON vRule.RuleGuid = vDisplayStringRule.ElementGuid

AND vDisplayStringRule.LanguageCode = @LanguageCode

The WHERE clause is similar to the one in the Availability report. It retrieves all the data that falls between the two dates specified in the date picker:

WHERE (vPerf.DateTime >= DATEADD(hh, DATEPART(hh, @StartDate), convert(varchar(8), @StartDate, 112)))

AND (vPerf.DateTime < DATEADD(hh, DATEPART(hh, @EndDate), convert(varchar(8), @EndDate, 112)))

The important points to note in this query are that the ManagedEntityRowId is a useful foreign key, and the format of the WHERE clause is similar to that used in the Availability report.

Here’s what the complete query looks like

SELECT

vPerf.DateTime,

vPerf.SampleCount,

vPerf.AverageValue,

vPerf.MinValue,

vPerf.MaxValue,

vPerf.StandardDeviation,

vPerformanceRuleInstance.RuleRowId,

vPerformanceRuleInstance.InstanceName,

vManagedEntity.ManagedEntityRowId,

vManagedEntity.ManagedEntityGuid,

vManagedEntity.ManagedEntityDefaultName,

vManagedEntity.Path,

ISNULL(vDisplayString.Name,vManagedEntityType.ManagedEntityTypeDefaultName)

AS DisplayName,

vManagementGroup.ManagementGroupGuid,

vManagementGroup.ManagementGroupDefaultName,

vRule.RuleGuid,

ISNULL(vDisplayStringRule.Name,vRule.RuleDefaultName) AS RuleDisplayName,

vPerformanceRule.MultiInstanceInd,

OptionList.[Group],

OptionList.GroupTitle,

OptionList.Position,

OptionList.ChartScale,

OptionList.ChartType,

OptionList.ChartColor,

OptionList.OptionXml,

vManagedEntityTypeImage.Image

FROM Perf.vPerfDaily as vPerf

INNER JOIN vPerformanceRuleInstance

ON vPerformanceRuleInstance.PerformanceRuleInstanceRowId = vPerf.PerformanceRuleInstanceRowId

INNER JOIN vManagedEntity

ON vPerf.ManagedEntityRowId = vManagedEntity.ManagedEntityRowId

INNER JOIN vManagedEntityType

ON vManagedEntityType.ManagedEntityTypeRowId = vManagedEntity.ManagedEntityTypeRowId

INNER JOIN vManagementGroup

ON vManagementGroup.ManagementGroupRowId = vManagedEntity.ManagementGroupRowId

INNER JOIN vRule

ON vPerformanceRuleInstance.RuleRowId = vRule.RuleRowId

INNER JOIN vPerformanceRule

ON vPerformanceRuleInstance.RuleRowId = vPerformanceRule.RuleRowId

INNER JOIN #InstanceList AS InstanceList

ON (vPerformanceRuleInstance.PerformanceRuleInstanceRowId = InstanceList.PerformanceRuleInstanceRowId)

INNER JOIN #OptionList As OptionList

ON (InstanceList.Position = OptionList.Position)

INNER JOIN #ObjectList AS ObjectList

ON (InstanceList.Position = ObjectList.Position)

AND (vManagedEntity.ManagedEntityRowId = ObjectList.ManagedEntityRowId)

LEFT OUTER JOIN vManagedEntityTypeImage

ON vManagedEntity.ManagedEntityTypeRowId = vManagedEntityTypeImage.ManagedEntityTypeRowId

AND vManagedEntityTypeImage.ImageCategory = N'u16x16Icon'

LEFT OUTER JOIN vDisplayString

ON vManagedEntityType.ManagedEntityTypeGuid = vDisplayString.ElementGuid

AND vDisplayString.LanguageCode = @LanguageCode

LEFT OUTER JOIN vDisplayString vDisplayStringRule

ON vRule.RuleGuid = vDisplayStringRule.ElementGuid

AND vDisplayStringRule.LanguageCode = @LanguageCode

WHERE (vPerf.DateTime >= DATEADD(hh, DATEPART(hh, @StartDate), convert(varchar(8), @StartDate, 112)))

AND (vPerf.DateTime < DATEADD(hh, DATEPART(hh, @EndDate), convert(varchar(8), @EndDate, 112)))

Creating Custom Queries

The core activity when creating a custom report is creating a custom Transact-SQL query to retrieve the information you need. When you design a custom report, if you are unfamiliar with Transact-SQL, it may be easier to obtain a Transact-SQL query from an outside source, such as [Kevin Holman’s OpsMgr Blog](http://go.microsoft.com/fwlink/?LinkId=207524) (http://go.microsoft.com/fwlink/?LinkId=207524) or [SQL Server Central](http://go.microsoft.com/fwlink/?LinkId=207525) (http://www.sqlservercentral.com/) rather than attempting to create an original query.

This section presents three examples of custom queries, including three different types of data: performance, alerts, and availability (or state). These examples should provide a foundation for creating custom queries of your own.

Performance Report

The first example creates a processor performance report for a particular server. A report from the generic library can provide the performance information but it may not be in the best grouping, format, or appearance.

This report includes performance data and the query should be run against the vPerfHourly view. vPerfDaily could also be used, depending on the desired level of detail of the data. In this example we have used vPerfHourly. Therefore, vPerfHourly is the view used in the FROM clause, with vPerf being the alias.

The next step is to determine the specific pieces of information from this dataset needed for the report. In this case, this is the DateTime, SampleCount, AverageValue, MinValue, MaxValue, and StandardDeviation columns from the vPerfHourly view. These columns are added to the SELECT clause.

As a best practice, when using the larger tables such as the vPerformance or vEvent tables, always specify date clauses in the WHERE clause. This ensures the use of the table index for fast querying.

The following is the first iteration of the query:

SELECT

vPerf.DateTime,

vPerf.SampleCount,

vPerf.AverageValue,

vPerf.MinValue,

vPerf.MaxValue,

vPerf.StandardDeviation

FROM Perf.vPerfHourly AS vPerf

WHERE

vPerf.DateTime >= '2010-06-28 15:19:59.387'

and vPerf.DateTime < '2010-07-10 15:19:59.387'

ORDER BY vPerf.DateTime

This query returns the requested data from the vPerfHourly table: date and time, Sample Count, Average, minimum value, maximum value, and standard deviation, within the times specified. However, the data is not filtered with regard to performance rule or managed instance; all the rows in the database between those two dates are retrieved.

The next step is to filter the results of the previous query to just the % Processor Time counter. The processor time counter requires the use of the Processor object. Therefore, the next version of the query requires the CounterName and ObjectName columns from the vPerformanceRule view. There is no foreign key to join vPerfHourly directly to vPerformanceRule. However, vPerformanceRule can be joined to vPerformanceRuleInstance using RuleRowId as a foreign key. In turn, vPerformanceRuleInstance can be joined to vPerfHourly using PerformanceRuleInstanceRowId. The modified query looks like this:

SELECT

vPerf.DateTime,

vPerf.SampleCount,

vPerf.AverageValue,

vPerf.MinValue,

vPerf.MaxValue,

vPerf.StandardDeviation,

vPerformanceRuleInstance.InstanceName,

vPerformanceRule.ObjectName,

vPerformanceRule.CounterName

FROM Perf.vPerfHourly AS vPerf INNER JOIN

vPerformanceRuleInstance ON vPerformanceRuleInstance.PerformanceRuleInstanceRowId = vPerf.PerformanceRuleInstanceRowId INNER JOIN

vPerformanceRule ON vPerformanceRuleInstance.RuleRowId = vPerformanceRule.RuleRowId

WHERE

vPerf.DateTime >= '2010-06-28 15:19:59.387'

and vPerf.DateTime < '2010-07-10 15:19:59.387'

AND (vPerformanceRule.ObjectName IN ('Processor'))

AND (vPerformanceRule.CounterName IN ('% Processor Time'))

ORDER BY vPerf.DateTime

This version of the query returns the data just for the % Processor Time counter. However, it is still reporting on every processor object in the database not just the specified entities.

The next step is to join the current query the objects to report on the managed servers. To do this, the query needs to join the vManagedEntity view in the FROM clause using ManagedEntityRowId as the foreign key. A specific ManagedEntityRowId is added to the WHERE clause for retrieval. In addition, vManagedEntity.Path is added to the SELECT clause. Here’s the next part of the query:

SELECT

vPerf.DateTime,

vPerf.SampleCount,

vPerf.AverageValue,

vPerf.MinValue,

vPerf.MaxValue,

vPerf.StandardDeviation,

vPerformanceRuleInstance.InstanceName,

vManagedEntity.Path,

vPerformanceRule.ObjectName,

vPerformanceRule.CounterName

FROM Perf.vPerfHourly AS vPerf INNER JOIN

vPerformanceRuleInstance ON vPerformanceRuleInstance.PerformanceRuleInstanceRowId = vPerf.PerformanceRuleInstanceRowId INNER JOIN

vManagedEntity ON vPerf.ManagedEntityRowId = vManagedEntity.ManagedEntityRowId INNER JOIN

vPerformanceRule ON vPerformanceRuleInstance.RuleRowId = vPerformanceRule.RuleRowId

WHERE

vPerf.DateTime >= '2010-06-28 15:19:59.387'

and vPerf.DateTime < '2010-07-10 15:19:59.387'

AND (vPerformanceRule.ObjectName IN ('Processor'))

AND (vPerformanceRule.CounterName IN ('% Processor Time'))

AND (vManagedEntity.Path IN ('OM10DC.OM10.lab'))

ORDER BY vPerf.DateTime

This query returns the results for the managed servers on the specified computer OM10DC.OM10.lab. This result is not particularly flexible because the dates, counters, and server are all coded into the query. Allowing the user to choose these parameters is described in [Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b).

Alert Report

The following sample query retrieves information about the alerts generated by the SQL Service Windows Service monitor in the month of June 2010. This query returns the path and name of the managed object, the Alert name, the time the alert was raised, the value of Parameter 1 in the Alert, how long it took for the alert to be resolved, and the name of the user who resolved it.

The query begins by selecting the desired information from the vAlert view:

SELECT

vAlert.AlertName,

vAlert.RaisedDateTime,

FROM

Alert.vAlert AS vAlert

ORDER BY RaisedDateTime

This query returns the two specified columns for all the alerts in the vAlert table. The ParameterValue column isn’t found in the vAlert view as it’s in the vAlertParameter view, which can be joined using AlertGuid as the foreign key. Likewise, the StateSetByUserId and TimeFromRaisedSeconds columns are found in the vAlertResolutionState view, which can also be joined using the AlertGuid as the foreign key. The query should only return alerts where the resolution state is “resolved” which is a value of 255. Therefore, that condition is added to the WHERE clause.

SELECT

vAlert.AlertName,

vAlert.RaisedDateTime,

vAlertParameter.ParameterValue,

vAlertResolutionState.TimeFromRaisedSeconds,

vAlertResolutionState.StateSetByUserId

FROM

Alert.vAlert AS vAlert

INNER JOIN Alert.vAlertParameter AS vAlertParameter

ON vAlertParameter.AlertGuid=vAlert.AlertGuid

INNER JOIN Alert.vAlertResolutionState AS vAlertResolutionState

ON vAlertResolutionState.AlertGuid=vAlert.AlertGuid

WHERE

vAlertResolutionState.ResolutionState=255

ORDER BY RaisedDateTime

The next step is to add the desired dates to the WHERE clause as well as the requirement that the alert has a value in parameter 1:

SELECT

vAlert.AlertName,

vAlert.RaisedDateTime,

vAlertParameter.ParameterValue,

vAlertResolutionState.TimeFromRaisedSeconds,

vAlertResolutionState.StateSetByUserId

FROM

Alert.vAlert AS vAlert

INNER JOIN Alert.vAlertParameter AS vAlertParameter

ON vAlertParameter.AlertGuid=vAlert.AlertGuid

INNER JOIN Alert.vAlertResolutionState AS vAlertResolutionState

ON vAlertResolutionState.AlertGuid=vAlert.AlertGuid

WHERE

vAlertResolutionState.ResolutionState=255

AND vAlertParameter.ParameterIndex=1

AND vAlert.RaisedDateTime between '2010-06-01' and '2010-06-30'

ORDER BY RaisedDateTime

The next step is to add the requirement that the alert is raised by the SQL Server Windows Service monitor. This is done by adding a restriction on MonitorDefaultName to the WHERE clause. MonitorDefaultName is in the vMonitor view, which can be joined to the vAlert view by setting vmonitor.MonitorRowId equal to vAlert.WorkflowRowId:

SELECT

vAlert.AlertName,

vAlert.RaisedDateTime,

vAlertParameter.ParameterValue,

vAlertResolutionState.TimeFromRaisedSeconds,

vAlertResolutionState.StateSetByUserId

FROM

Alert.vAlert AS vAlert

INNER JOIN Alert.vAlertParameter AS vAlertParameter

ON vAlertParameter.AlertGuid=vAlert.AlertGuid

INNER JOIN Alert.vAlertResolutionState AS vAlertResolutionState

ON vAlertResolutionState.AlertGuid=vAlert.AlertGuid

INNER JOIN vMonitor ON vMonitor.MonitorRowId=vAlert.WorkflowRowid

WHERE

vAlertResolutionState.ResolutionState=255

AND vAlertParameter.ParameterIndex=1

AND vAlert.RaisedDateTime between '2010-06-01' and '2010-06-30'

AND vMonitor.MonitorDefaultName='SQL Server Windows Service'

ORDER BY RaisedDateTime

The final iteration of the query adds the name and path of the managed entity to the SELECT statement. This requires a join to the vManagedEntity view, which can be joined to the vAlert view using ManagedEntityRowId as the foreign key. There is also an addition to the WHERE clause to restrict results to rows where the type of the managed entity is SQL DB Engine. This requires a join to the ManagedEntityType view, which can be joined to the vManagedEntity view on vManagedEntityTypeRowId:

SELECT

vManagedEntity.Path,

vManagedEntity.Name,

vAlert.AlertName,

vAlert.RaisedDateTime,

vAlertParameter.ParameterValue,

vAlertResolutionState.TimeFromRaisedSeconds,

vAlertResolutionState.StateSetByUserId

FROM

Alert.vAlert AS vAlert

INNER JOIN vManagedEntity ON vManagedEntity.ManagedEntityRowId=vAlert.ManagedEntityRowId

INNER JOIN vManagedEntityType ON vManagedEntityType.ManagedEntityTypeRowId=vManagedEntity.ManagedEntityTypeRowId

INNER JOIN Alert.vAlertParameter AS vAlertParameter

ON vAlertParameter.AlertGuid=vAlert.AlertGuid

INNER JOIN Alert.vAlertResolutionState AS vAlertResolutionState

ON vAlertResolutionState.AlertGuid=vAlert.AlertGuid

INNER JOIN vMonitor ON vMonitor.MonitorRowId=vAlert.WorkflowRowid

WHERE

vAlertResolutionState.ResolutionState=255

AND vAlertParameter.ParameterIndex=1

AND vAlert.RaisedDateTime between '2010-06-01' and '2010-06-30'

AND vMonitor.MonitorDefaultName='SQL Server Windows Service'

AND vManagedEntityType.ManagedEntityTypeDefaultName ='SQL DB Engine'

ORDER BY RaisedDateTime

This final version of the query returns all the resolved alerts generated by the SQL Server Windows Service during the month of June 2010 as the requirements stated. As with the performance report, this report is not very flexible because the dates, resolution state, and managed entity cannot be modified without editing the Transact-SQL query directly. These elements can be changed with parameters as described in [Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b)

Availability Report

The following sample query will retrieve information about when the Computer Availability Health Rollup monitor was in the red or the yellow state for the month of June, 2010. The generic availability report will provide the information for all states and for specific objects, but this custom report has a narrower focus. Availability information is found in the state views, specifically vStateDaily and vStateHourly. This query uses vStateDaily.

The first iteration of the query retrieves the time in the red and yellow states together with the time stamp that the state was retrieved for the month of June 2010.

SELECT

vStateDaily.DateTime,

vStateDaily.InRedStateMilliseconds,

vStateDaily.InYellowStateMilliseconds

FROM

State.vStateDaily AS vStateDaily

WHERE

DateTime BETWEEN '2010-06-01' and '2010-06-30'

ORDER BY DateTime

The next step is to specify the monitor in order to narrow the returned data. The monitors are found in the vMonitor view, which cannot be joined to vStateDaily, but can be joined to vManagedEntityMonitor using MonitorRowId as the foreign key. vManagedEntityMonitor can then be joined to vStateDaily using ManagedMonitorRowId as a foreign key.

SELECT

vStateDaily.DateTime,

vStateDaily.InRedStateMilliseconds,

vStateDaily.InYellowStateMilliseconds

FROM

State.vStateDaily AS vStateDaily

INNER JOIN vManagedEntityMonitor ON vManagedEntityMonitor.ManagedEntityMonitorRowId=vStateDaily.ManagedEntityMonitorRowId

INNER JOIN vMonitor ON vMonitor.MonitorRowId=vManagedEntityMonitor.MonitorRowId

WHERE

vMonitor.MonitorDefaultName='Computer Availability Health Rollup'

AND DateTime BETWEEN '2010-06-01' and '2010-06-30'

ORDER BY DateTime

The next step is to retrieve the path and name for the managed entity. As in the Alert monitor section, these columns are found in the vManagedEntity view, which can be joined to vManagedEntityMonitor using MonitorRowId as the foreign key.

SELECT

vManagedEntity.Path,

vManagedEntity.Name,

vStateDaily.DateTime,

vStateDaily.InRedStateMilliseconds,

vStateDaily.InYellowStateMilliseconds

FROM

State.vStateDaily AS vStateDaily

INNER JOIN vManagedEntityMonitor ON vManagedEntityMonitor.ManagedEntityMonitorRowId=vStateDaily.ManagedEntityMonitorRowId

INNER JOIN vMonitor ON vMonitor.MonitorRowId=vManagedEntityMonitor.MonitorRowId

INNER JOIN vManagedEntity ON vManagedEntity.ManagedEntityRowId=vManagedEntityMonitor.ManagedEntityRowId

WHERE

vMonitor.MonitorDefaultName='Computer Availability Health Rollup'

AND DateTime BETWEEN '2010-06-01' and '2010-06-30'

ORDER BY DateTime

As with the other two reports, the monitor and dates are coded directly into the query and are not flexible. This can be solved by using parameters as described in [Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b)

Using a Report Builder

After you have a query that retrieves the data that you need, you are ready to create a report. There are several tools that can be used to create a report: Microsoft Business Intelligence Development Studio (BIDS), Microsoft Visual Studio, Report Builder, or Microsoft Excel. This guide uses BIDS because it is already installed with Microsoft SQL Server and integrates seamlessly with Microsoft SQL Server Reporting Services (SSRS).

Creating a New Report Project

The easiest way to create a new report is to use the report wizard, which creates a simple report and populates it with a basic data display that you select. You can also create a report server project, which creates a blank report project that you then populate with a report and the data that you want to display.

To create a new report project by using the report wizard

|  |
| --- |
| 1. Open BIDS, click File, click New, and then click Project.  2. In the New Project window, in the Project Types pane, make sure that Business Intelligence Projects is selected. In the Templates pane, select Report Server Project Wizard. Enter a name for the project, and then click OK.  3. The Report Wizard starts. On the welcome page, click Next.  4. On the Select the Data Source page, enter DataWarehouseMain as the name, leave the Type as Microsoft SQL Server, and make sure that you select the Make this a shared data source check box. Click the Edit button.  5. The Connection Properties dialog box appears. In the Server Name field, enter the name of the server. In the Log on to the server box, enter the appropriate credentials, as discussed in [Setting up the Environment](#z50474d3c3da644c480600afc8cb7863d). In the Connect to a database box, select the Select or enter a database name option. In the drop-down box, select the OperationsManagerDW database, and then click OK. The wizard will reappear with the Connection String box filled in. Click Next.  6. The Design the Query page of the wizard appears. In the Query string box, if you have created the query already, as described in [Creating Custom Queries](#zc41f0cb483d04c5085fa2bfd935c23e3), paste the query into the Query String field. If you have not yet created a query, click the Query Builder button to start the Query Designer.  7. Within the Query Designer, the default view is the graphical query designer. For more information about how to design a query visually, see [the guide to Query Designer](http://go.microsoft.com/fwlink/?LinkId=207899) (http://go.microsoft.com/fwlink/?LinkID=207899). Click the Edit As Text button to edit the query directly in the window. Click the Import… button to use a query that you saved previously. The ! button runs the query and displays the results in the lower pane of the Query Designer. If the report does not show the data that you expected, you can revise the query and test it without recreating the report. After you have created the query, click OK.  8. You are returned to the Design the Query page of the wizard. Click Next.  9. On the Select the Report Type page of the wizard, select Tabular or Matrix. A tabular report returns columns; a matrix report uses an intersection of columns and rows for a more precise view of the data. Click Next.  10. On the Design the Table page of the wizard, select how you want the data in the table to be organized. The options are as follows:   The Page option indicates the fields that appear at the page level of the report.   The Group option indicates the fields by which to group the data in the table.   The Details option indicates the fields that are displayed in the details section of the table.  To change the order of the fields in any group, select a field, and then click the up button or the down button. You can also accept the defaults by clicking Next.  11. On the Choose the Table Style page of the wizard there is a list of several styles that affect color and layout choices. Select the visual style you prefer for the table. Click Next.  12. On the Choose the Deployment Location page of the wizard, select where the report will be deployed. By default, the URL for the Operations Manager reporting server is entered. However, you should confirm that this URL is correct. Enter a name for the Deployment folder or accept the default.  13. On the Completing the Wizard page, enter a name for the report. This will be displayed on the report page. You can review the report summary data on this page. If you want to adjust any of the settings, click the Back button and make the appropriate changes. If you select the Preview report check box, the editor will open in the Preview tab instead of the Design tab. Click Finish to continue.  14. The basic layout of the report appears in the main window of BIDS. You will see the following two tabs:   The Design tab, which shows the layout of the report with the fields returned from the query in place   The Preview tab, which shows what the data looks like in the report |

To create a report without using the report wizard

|  |
| --- |
| 1. Open BIDS, click File, and then click New Project.  2. In the New Project window, in the Project Types pane, make sure that Business Intelligence Projects is selected. In the Templates pane, select Report Server Project. Enter a name for the project, and then click OK.  3. The project is created, but still shows the Start page as there is no report in the project yet. In the menu bar, click Project, and then click Add New Item. The Add New Item dialog box appears. In the Templates pane, select Report, and then click Add. The design surface appears in the main window with the Design tab selected, which shows a blank report.  4. You must add a data source and a dataset to your report in order to display any data. In the Report Data pane, click New, and then click Data Source. The Data Source Properties dialog box appears. Provide a name for the data source (DataWarehouseMain is recommended) or accept the default. If you have not created a shared data source, select Embedded connection; for the Type, select Microsoft SQL Server, and for the Connection string enter the connection information for the OperationsManagerDW database. If you created a shared data source previously, as discussed in [Setting up the Environment](#z50474d3c3da644c480600afc8cb7863d), select Use shared data source reference, and then select the shared data source in the box. Click OK.  5. In the Report Data pane, click New, and then click Dataset. Enter a name for the dataset or use the default. Select the Data source, usually DataWarehouseMain, and then enter the query. This can either a query created earlier, or click the Query Designer button and create the query using the Graphical Query Designer. When you have finished, click OK.  6. The dataset appears in the Report Data window with the available dataset fields retrieved by the query displayed below the dataset name.  7. Click View, and then click Toolbox to display the Toolbox dialog box. The Toolbox contains several standard items that can be used in reports. From the Toolbox, select a Table, Matrix, Chart, or Gauge, depending on how you want to represent the data, and drag it onto the design surface of the report. From the Report Data window, drag the desired fields onto the Table, Matrix, Chart, or Gauge tool, as appropriate.  8. Check the appearance of the report by clicking the Preview tab. |

The following sections discuss how to change specific aspects of the report. For more information about report design, see [SQL Server 2008 Books Online](http://go.microsoft.com/fwlink/?LinkId=207889)(http://go.microsoft.com/fwlink/?LinkID=207899).

Changing the Data in the Report

To change the data that’s displayed in the report, you must change the query.

To change the query

|  |
| --- |
| 1. In the Report Data tree, right-click the dataset, by default this is DataSet1, and then select Query. This starts the Query Designer.  2. Within the Query Designer, click the Edit As Text button to edit the query directly in the window. Click the Import… button to use a query that you saved previously. The ! button runs the query and displays the results in the lower pane of the Query Designer. If the report does not show the data that you expected, you can revise the query and test it without recreating the report. |

Changing the Appearance of the Report

Within the report dataset, you can change the rows, columns, or individual cells. When you click any cell in the dataset, borders will appear around the table. You can click the border to select a row or a column. To select a specific cell in the table, click in that cell. The properties for the row, column, or cell you selected are displayed in the Properties window. By default, this window is located at the lower-right corner of the interface.

To change the name of a column, click in the column heading and edit the text in the text box. You can also change the font, background color, text color, and many other properties of the text box with the Properties window. Drag the borders of the column to change the column’s size. Reorder the columns by dragging them to the new desired position.

To enable sorting in table columns

|  |
| --- |
| 1. Right-click the column to which you want to add sorting, and click Text Box Properties. The Text Box Properties window opens.  2. Click the Interactive Sorting tab. Select the check box Enable interactive sorting on this text box. You can select to sort by detail rows or by groups. Select the column to sort by, and then click OK. |

Note: If you want the report to sort on a particular field by default, add an ORDER BY clause to the query.

Adding a Chart to the Report

Instead of, or in addition to, the table or matrix you may want to include a chart in the report for a visual representation of the data.

To add a chart to the report

|  |
| --- |
| 1. You may need to expand the report area to provide sufficient room for the chart by using the resize handles.  2. Click View, and then click Toolbox. The toolbox dialog box opens; you may want to pin the toolbox in place. In the toolbox, click Chart, and then click in the report to place the chart, or drag to size the chart in the report space.  3. The Select Chart Type dialog box appears, giving you various chart choices. Select the type of chart you want to add, and then click OK. The chart appears on the design surface.  4. Drag the chart in the report to move it or use the resize controls to resize it.  5. Click the area labeled Axis Title to change the text on the axis.  6. To add data fields to the report, select the Report Data pane (click View, and then Report Data if the Report Data pane is no longer visible), and expand the dataset to show the data fields the query retrieves. On the design surface, click the chart that you added. This reveals three areas around the border of the chart as follows:  a. Drop data fields here: Drag data fields here that you want to represent as values in the chart (for example, the height of the bars in a bar graph).  b. Drop category fields here: Drag data fields here for which you want to present the values (for example, the items on the x-axis in a bar graph).  c. Drop series fields here: These fields are optional. Drag data fields here to add an extra dimension to the chart (for example, if a bar graph has multiple data fields for a single category field).  7. With the chart selected, the Properties window lets you define the properties of the chart itself, such as the colors, borders, and size. |

For more information about charts, see [Working with Chart Data Regions](http://go.microsoft.com/fwlink/?LinkId=207928) in SQL Server Books Online: http://go.microsoft.com/fwlink/?LinkID=207928.

Adding an Image to the Report

You may want to add a static image to the report, for example, a company or department logo.

To add an image to the report

|  |
| --- |
| 1. Open the Toolbox, click the Image item, and drag it onto the design area. The Image Properties dialog box appears.  2. In the Image Properties dialog box, provide a name for the image (which will not be displayed), and an optional tooltip (displayed when the users hovers the pointer over the image). The Select the image source box has the following three options:   External: For images that exist on the report server or another external source. If you select this option, you must provide a URL to the image location in the Use this image field. If the image is changed at its source, the image in the report will change dynamically.   Embedded: For images that do not have to be changed. Click the Import button and browse to the image location. The image is embedded in the report and will not change dynamically.   Database: For images stored in the database. Provide the path of the image in the Use this field box. In the Use this MIME type box, select the appropriate MIME type for the image.  3. The Size, Visibility, Action, and Border tabs in the Image Properties dialog box enable you to change the appearance of the image in the report, and also enable you to set the image as a link to another report or URL. |

For more information about how to use images in reports, see [Images](http://go.microsoft.com/fwlink/?LinkId=207938) in SQL Server Books Online (http://go.microsoft.com/fwlink/?LinkID=207938)

Adding an Additional Dataset to the Report

You may want to add another set of data to the same report, using a different dataset. This lets you consolidate reports into a single document for ease of reference.

To add an additional dataset to the report

|  |
| --- |
| 1. In the Report Data window, click New, and then click Dataset. The Dataset Properties dialog box appears.  2. In the Name field, enter a name for the dataset or use the default.  3. In the Data source field, select the data source, usually DataWarehouseMain.  4. In the Query field, enter the query. This can either a query created earlier or click the Query Designer button to create a query. When you have finished, click OK.  5. A second dataset will appear in the Report Data window. From the Toolbox, select a Table, Matrix, Chart, or Gauge, depending on how you want to represent the data. Drag the selected control onto the design surface, and a new data item will appear in the report.  6. From the Report Data window, drag the desired data fields onto the Table, Matrix, Chart, or Gauge tool, as appropriate. |

Custom Report Parameters

The Transact-SQL queries described in [Creating Custom Queries](#zc41f0cb483d04c5085fa2bfd935c23e3) can retrieve a wide variety of data from the OperationsManagerDW database, but the reports are limited to a specific set of objects at a specific time. To offer flexibility in the data that is reported, you can use parameters to let users define the terms of the report.

When you create a report in Microsoft SQL Server Business Intelligence Development Studio (BIDS), a number of parameters are available to use in your reports.

In addition to BIDS, Operations Manager 2007 R2 includes the smart parameter block to enable you to easily offer controls for selecting parameters that are intuitive to report users, such as a date/time picker that can be limited to business hours that you define, or an object picker that makes it easy to select specific objects in Operations Manager.

To use the smart parameter block, you must first include the report in a management pack, and then import the management pack into Operations Manager, as described in [Deploying Reports](#z18c5f1eb721f45209918d45203a3ef7f). After the report has been deployed, you can edit the parameter block with the Authoring console.

Note

The Authoring console is available as part of the [System Center Operations Manager Resource Kit](http://go.microsoft.com/fwlink/?LinkID=192098). For more information about the Authoring console, see the Introduction to the Authoring Console.

Using Parameters in BIDS

If you do not require the enhanced controls in the smart parameter block, you can create parameters for your report by using the parameter tools in Business Intelligence Development Studio.

To add parameters in BIDS

|  |
| --- |
| 1. Open the report in BIDS.  2. BIDS automatically adds parameters to your report based on the parameters that are used in the query. To edit the query, right-click the dataset for your report (by default, DataSet1) and select Query. The Query Designer opens.  3. In the query, edit the FROM clause to include the parameters that you want the user to select in the report. Preface each parameter with an @ symbol. The following code is an example of a query with hard-coded dates.  vPerf.DateTime >= '2010-06-28 15:19:59.387'  and vPerf.DateTime < '2010-07-10 15:19:59.387'  If you replace those hard-coded dates with parameters, the query resembles the following code.  vPerf.DateTime >= @StartDate  and vPerf.DateTime < @EndDate  4. Click the ! button to run the query. The Query Designer opens a Define Query Parameters window. You do not have to enter values for the parameters at this time, unless you want to test the query. Click OK to close the Define Query Parameters window, and click OK to close the Query Designer.  5. BIDS has automatically created parameters as part of the report. However, you must define the data type of the parameters so that BIDS can provide the correct controls in the report. In the Report Data pane, expand Parameters. The parameters that you added to your query appear. Double-click one of the parameters, for example, @StartDate, to open the Report Parameter Properties dialog box for that parameter. On the General tab of this dialog box, you can set basic information about this parameter. Most of the settings are optional, but name and data type are required.   Name: The name of the parameter that you entered in the query. This field will be populated for you.   Prompt: The text that will be displayed in the report next to the control that sets this parameter. Enter whatever prompt makes sense for your report users, for example “Start date” instead of “StartDate”.   Data type: Use the drop-down list to select the type of data that this parameter represents.   Text: For plain text data. This type is the default, and the report generates a standard text entry field for this parameter.   Boolean: For true/false values. The report generates a check box for this parameter.   Date/Time: For date and time values. The report generates a text field with a calendar control for this parameter.   Integer: For whole numbers. The report generates a text field for this parameter.   Float: For decimal numbers. The report generates a text field for this parameter.   The next three check boxes let you select whether blank values are permitted, or null values, or multiple values. Selecting Allow multiple values causes the report to render the control for this parameter as a drop-down list.   Select the visibility of the parameters (Visible, Hidden, or Internal). Most of the time, you should select Visible, so the report users can use this control. You might select Hidden or Internal if you intend to calculate the value of the parameter from other data, but do not want the report user to have direct access.  6. Select the Available Values tab if you want to restrict the values that the report user can enter. For more information, see [Creating Report Parameters and Setting Report Parameter Properties](http://go.microsoft.com/fwlink/?LinkId=220277) (http://go.microsoft.com/fwlink/?LinkId=220277).  7. Select the Default Values tab if you want to prepopulate this control with default data. For more information, see [Creating Report Parameters and Setting Report Parameter Properties](http://go.microsoft.com/fwlink/?LinkId=220277) (http://go.microsoft.com/fwlink/?LinkId=220277).  8. Select the Advanced tab to specify whether the report immediately refreshes when the value of the control changes. By default, the report automatically determines when to refresh.  9. Click OK when you have finished setting the properties for this parameter. Set the properties for any other parameters used in this report.  10. Run the report by selecting the Preview tab in BIDS. The controls that you configured appear at the top of the report. You can enter data for these parameters, and click View Report to see a preview of the results.  11. Save your report when you are finished. |

Adding Parameters in the Smart Parameter Block

The smart parameter block lets you use several tools that are not available through BIDS, such as the relative date-time picker. To use the smart parameter block, you must use the Authoring console, and also edit the XML of your management pack directly. For information about the controls that are available, see Report Controls. However, linked reports involve creating a report from an existing base report. Custom reports have no base report and therefore require more code for the smart parameter block to work.

To add parameters in the smart parameter block

|  |  |
| --- | --- |
| 1. Open the management pack in the Authoring console.  2. Click the Reporting tab, and then click Reports in the Reporting pane. The reports that are part of the management pack appear in the results pane.  3. Right-click the report that you want to edit, and then select Properties. The Properties window for the report appears.  4. Click the Parameter Block tab. The Parameter Block field is most likely empty. Click Edit in in external editor. The button starts the external editor, and opens the .rpdl file that is a part of the management pack.  5. Decide on the parameters that you want for this report. For information about which parameters are available, see Report Parameters.  6. Locate the XML code for the control that you want to use in Report Controls and its subsections. Copy that XML into the external editor. Make a note of the ReportParameter name element for the control that you use, because you have to copy it to the query in the report definition.  7. Add a Controls tag to enclose the Control tag that you just added, as the following example shows.  <Controls>  <Control…>  …  </Control>  </Controls>  8. Add a ParameterBlock tag to enclose the Controls tag. The columns property of this tag defines the width of the parameter block in the report. The xmlns property of this tag defines the namespace for this element, and must be set to http://schemas.microsoft.com/mom/reporting/2007/ReportParameterSettings. The XML for the parameter block should now look like this:  <ParameterBlock columns="6" xmlns="http://schemas.microsoft.com/mom/reporting/2007/ReportParameterSettings">  <Controls>  <Control…>  …  </Control>  </Controls>  </ParameterBlock>  9. Save the file, and then close the external editor. The XML you just entered appears on the Properties window.  10. You now must edit the query in the report definition. Click the Definition tab to see the .rpdl file for the report. You can edit the query directly on this window, or click Edit in external editor to edit the file. However, editing the report in BIDS instead of the external editor makes the next step easier.  To edit the query in BIDS   |  | | --- | | a. Open the report project that contains the report that you want to edit.  b. In the Report Data pane, right-click the dataset to select Query to open the Query Designer window.  c. The control you want to use dictates how the parameter will be used in the query, but the parameter will always be used in the WHERE clause. In the editor, delete the value that you want to replace with the parameter (if any), and insert the ReportParameter name associated with the control from the .rpdl file. If the parameter can have a single value, preface the name with an @ character. You must use the exact report parameter name, or the query cannot associate the query parameter with the control name in the parameter block.  d. Click OK to save the revised query, and then save the report in BIDS.  e. You now must add the revised report definition to the management pack. In the Authoring console, open the Properties window for the report if it is not already open, click the Definition tab, click Load content from file, and then select the .rdl file that you just saved in BIDS. The code now appears in the RDL field.  f. Delete the line that begins:  <?xml version |   11. You now must add a ReportParameters element to the report definition. This is separate from the ReportParameters section for the parameter block. In the Properties window for the report, click the Definition tab, and then click the Edit in External Editor button. The report opens in the XML editor of your choice. Following the closing /DataSources tag in the report, add the following code:  <ReportParameters>  <ReportParameter Name=" ">  <DataType></DataType>  <Prompt></Prompt>  </ReportParameter>  </ReportParameters>  The code used in the ReportParameters section depends on the control you choose. Open the .xml file for the Microsoft Generic Report Library and search for a report that uses the control that you want to use. Copy the report parameters section for that report from the Generic Report Library and paste it into your report.  If you edited the query in BIDS, and a ReportParamters section already exists in the report, delete it, and then replace it with the ReportParameters section you selected from the Generic Report Library.  12. You must now add the query parameters to the DataSet for this report. Following the /CommandText tag that contains the query for the report, add the following code:  <QueryParameters>  <QueryParameter Name="">  <Value></Value>  </QueryParameter>  </QueryParameters>  You must define a QueryParameter section for each parameter used in the query. If you edited the query in BIDS, and the QueryParameter section already exists, you only have to replace the Value elements. In the Name property, provide the name of the parameter used in the query. In the Value element, you must provide the appropriate function to define the value for this parameter. The function differs depending on the value you want to obtain; each function is defined in the code block that you will add next.  13. The smart parameter block requires additional code to process the report parameters. After the closing /DataSets tag, add the following Code element, and insert the following code:  <Code>  Const TimeZoneParameterName As String = "TimeZone"  Const SD\_BaseTypeParameterName As String = "StartDate\_BaseType"  Const SD\_BaseValueParameterName As String = "StartDate\_BaseValue"  Const SD\_OffsetTypeParameterName As String = "StartDate\_OffsetType"  Const SD\_OffsetValueParameterName As String = "StartDate\_OffsetValue"  Const ED\_BaseTypeParameterName As String = "EndDate\_BaseType"  Const ED\_BaseValueParameterName As String = "EndDate\_BaseValue"  Const ED\_OffsetTypeParameterName As String = "EndDate\_OffsetType"  Const ED\_OffsetValueParameterName As String = "EndDate\_OffsetValue"  Const IsRelativeTimeSupported As Boolean = False  Const TimeTypeParameterName As String = "TimeType"  Const TimeWeekMapParameterName As String = "TimeWeekMap"  Dim ReportTimeZone As Microsoft.EnterpriseManagement.Reporting.TimeZoneCoreInformation  Dim ReportStartDate As DateTime  Dim ReportEndDate As DateTime  Dim ReportTime As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTime  Dim ReportCulture As System.Globalization.CultureInfo  Dim ParameterProcessor As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor  Protected Overrides Sub OnInit()  ReportTimeZone = Nothing  ReportStartDate = DateTime.MinValue  ReportEndDate = DateTime.MinValue  ReportTime = Nothing  ReportCulture = System.Globalization.CultureInfo.GetCultureInfo(Report.User("Language"))  ParameterProcessor = New Microsoft.EnterpriseManagement.Reporting.ParameterProcessor(ReportCulture)  End Sub  Public Function GetCallingManagementGroupId() As String  Return Microsoft.EnterpriseManagement.Reporting.ReportingConfiguration.ManagementGroupId  End Function  Public Function GetReportTimeZone() As Microsoft.EnterpriseManagement.Reporting.TimeZoneCoreInformation  If IsNothing(ReportTimeZone) Then ReportTimeZone = Microsoft.EnterpriseManagement.Reporting.TimeZoneCoreInformation.FromValueString(Report.Parameters(TimeZoneParameterName).Value)  Return ReportTimeZone  End Function  Public Function ToDbDate(ByVal DateValue As DateTime) As DateTime  return GetReportTimeZone.ToUniversalTime(DateValue)  End Function  Public Function ToReportDate(ByVal DateValue As DateTime) As DateTime  return GetReportTimeZone.ToLocalTime(DateValue)  End Function  Public Function GetReportStartDate() As DateTime  If (ReportStartDate = DateTime.MinValue) Then  If (IsRelativeTimeSupported) Then  ReportStartDate = ParameterProcessor.GetDateTime(ToReportDate(DateTime.UtcNow), Report.Parameters(SD\_BaseTypeParameterName).Value, Report.Parameters(SD\_BaseValueParameterName).Value, Report.Parameters(SD\_OffsetTypeParameterName).Value, Report.Parameters(SD\_OffsetValueParameterName).Value, Report.Parameters(TimeTypeParameterName).Value)  Else  ReportStartDate = ParameterProcessor.GetDateTime(ToReportDate(DateTime.UtcNow), Report.Parameters(SD\_BaseTypeParameterName).Value, Report.Parameters(SD\_BaseValueParameterName).Value, Report.Parameters(SD\_OffsetTypeParameterName).Value, Report.Parameters(SD\_OffsetValueParameterName).Value)  End if  End If  return ReportStartDate  End Function  Public Function GetReportEndDate() As DateTime  If (ReportEndDate = DateTime.MinValue) Then  If (IsRelativeTimeSupported) Then  ReportEndDate = ParameterProcessor.GetDateTime(ToReportDate(DateTime.UtcNow), Report.Parameters(ED\_BaseTypeParameterName).Value, Report.Parameters(ED\_BaseValueParameterName).Value, Report.Parameters(ED\_OffsetTypeParameterName).Value, Report.Parameters(ED\_OffsetValueParameterName).Value, Report.Parameters(TimeTypeParameterName).Value)  If IsBusinessHours(GetReportTimeFilter()) Then ReportEndDate = ReportCulture.Calendar.AddDays(ReportEndDate, 1)  Else  ReportEndDate = ParameterProcessor.GetDateTime(ToReportDate(DateTime.UtcNow), Report.Parameters(ED\_BaseTypeParameterName).Value, Report.Parameters(ED\_BaseValueParameterName).Value, Report.Parameters(ED\_OffsetTypeParameterName).Value, Report.Parameters(ED\_OffsetValueParameterName).Value)  End if  End If  return ReportEndDate  End Function  Public Function GetReportTimeFilter() As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTime  If IsNothing(ReportTime) Then ReportTime = New Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTime(Report.Parameters(TimeTypeParameterName).Value, Report.Parameters(SD\_BaseValueParameterName).Value, Report.Parameters(ED\_BaseValueParameterName).Value, CStr(Join(Report.Parameters(TimeWeekMapParameterName).Value, ",")))  return ReportTime  End Function  Public Function IsBusinessHours(Value As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTime) As Boolean  return (Not IsNothing(Value)) And (Value.TimeType = Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTimeType.Business)  End Function  Public Function FormatDateTime(Format As String, Value As DateTime) As String  return Value.ToString(Format, ReportCulture)  End Function  Public Function FormatNumber(Format As String, Value As Decimal) As String  return Value.ToString(Format, ReportCulture)  End Function  Public Function FormatString(Format As String, ParamArray Values() as Object) As String  return String.Format(ReportCulture, Format, Values)  End Function  Public Function NullFormatString(Format As String, Value as String) As String  return IIF(String.IsNullOrEmpty(Value), String.Empty, String.Format(ReportCulture, Format, Value))  End Function  Public Function FormatBusinessHours(Format As String, Value As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTime) As String  Dim result As String  If IsBusinessHours(Value) Then    Dim firstDay As DayOfWeek  Dim days As System.Collections.Generic.List(Of String)  firstDay = ReportCulture.DateTimeFormat.FirstDayOfWeek  days = new System.Collections.Generic.List(Of String)()  For loopDay As DayOfWeek = DayOfWeek.Sunday To DayOfWeek.Saturday  Dim day As DayOfWeek  day = CType((CInt(loopDay) + CInt(firstDay)) Mod 7, DayOfWeek)  If value.WeekMap.Contains(day) Then days.Add(ReportCulture.DateTimeFormat.GetAbbreviatedDayName(day))  Next loopDay  result = FormatString(Format, DateTime.Today.Add(Value.StartTime).ToString(ReportCulture.DateTimeFormat.ShortTimePattern), DateTime.Today.Add(Value.EndTime).ToString(ReportCulture.DateTimeFormat.ShortTimePattern), String.Join(",", days.ToArray()))  Else  result = String.Empty  End if    return result  End Function  Public Function BuildXmlValueList(ByVal ValueList() As Object) As String  Return Microsoft.EnterpriseManagement.Reporting.MultiValueParameter.ToXml("Data", "Value", ValueList)  End Function  </Code>  This is the entire code section from the System Center Report Library, and contains more code than you require for a single control. However, including this code block guarantees that you have included the code you require for your chosen control.  14. After the /Code tag, add the following code:  <CodeModules>  <CodeModule>Microsoft.EnterpriseManagement.Reporting.Code, Version=6.0.0.0, Culture=neutral</CodeModule>  </CodeModules>  This element is required to enable the code in the Code section to work correctly.  15. Save the report definition in the external editor. The report definition now appears in the Properties window for the report in the RDL field. Delete the first line in that field, which looks like this:  <?xml version="1.0" encoding="utf-8"?>  Click OK to close the Properties window. Save the management pack.  16. Finally, you must add a reference to the System Center Data Warehouse Report Library. This cannot be done from within the Authoring console. Locate the file where your management pack is saved, and open it with the XML editor of your choice. Within the References section of the file, add the following code:  <Reference Alias="MicrosoftSystemCenterDataWarehouseReportLibrary">  <ID>Microsoft.SystemCenter.DataWarehouse.Report.Library</ID>  <Version>6.1.7221.0</Version>  <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>  </Reference>  17. Save the management pack, and then deploy it as discussed in [Deploying Reports](#z18c5f1eb721f45209918d45203a3ef7f). |

Example: Adding a Relative Date Time Picker to the Alert Report

In this example, you’ll add a Relative Date Time Picker control to the smart parameter block of the Alert Report created in Creating Cutom Queries ([Creating Custom Queries](#zc41f0cb483d04c5085fa2bfd935c23e3))

To add a custom control to the Alert Report

|  |
| --- |
| 1. Open the management pack containing the Alert report in the Authoring console. Select the Reporting tab, right-click the Alert report, and open its Properties window.  2. Click the Parameter Block tab, and click Edit in external editor. The external editor of your choice opens, showing a blank .rpdl file.  3. To add a Relative Date Time Picker control, find the XML code for the control in (Date Report Controls). The code in question looks like this:  <Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.RelativeDateTimePicker" columnSpan="2" rowSpan="2">  <ReportParameters>  <ReportParameter name="TimeZone" binding="TimeZone">  <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.TimeZone</Prompt>  </ReportParameter>  <ReportParameter name="TimeZoneName" binding="TimeZoneName" />  <ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />  <ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue">  <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>  </ReportParameter>  <ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />  <ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />  <ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />  <ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">  <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>  </ReportParameter>  <ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />  <ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />  </ReportParameters>  </Control>  4. Copy this code into your blank .rpdl file.  5. You also must add the ParameterBlock and Controls tags to enclose the Control tag. With the parameter block tags added, the code looks like this:  <ParameterBlock columns="6" xmlns="http://schemas.microsoft.com/mom/reporting/2007/ReportParameterSettings">  <Controls>  <Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.RelativeDateTimePicker" columnSpan="2" rowSpan="2">  <ReportParameters>  <ReportParameter name="TimeZone" binding="TimeZone">  <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.TimeZone</Prompt>  </ReportParameter>  <ReportParameter name="TimeZoneName" binding="TimeZoneName" />  <ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />  <ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue">  <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>  </ReportParameter>  <ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />  <ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />  <ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />  <ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">  <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>  </ReportParameter>  <ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />  <ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />  </ReportParameters>  </Control>  </Controls>  </ParameterBlock>  6. Save the file, and then close the external editor. The code now appears in the Parameter Block field.  7. Now you must change the query in the report definition to use the parameters. Open the report file in BIDS.  8. Right-click DataSet1 to select Query. The Query Designer opens, displaying the query for this dataset.  9. In the Query Designer, edit the WHERE clause, remove the hard-coded dates, and replace them with the parameters @StartDate and @EndDate. The additional CONVERT functions are necessary to convert the parameters to the DATETIME type. The WHERE clause of the query now looks like this:  WHERE (vAlertResolutionState.ResolutionState = 255) AND (vAlertParameter.ParameterIndex = 1)  AND (  CONVERT(DATETIME, CONVERT(VARCHAR, vAlert.RaisedDateTime, 101))  BETWEEN  CONVERT(DATETIME, CONVERT(VARCHAR, @StartDate, 101))  AND  CONVERT(DATETIME, CONVERT(VARCHAR, @EndDate, 101))  )  AND (vMonitor.MonitorDefaultName = 'Availability') AND (vManagedEntityType.ManagedEntityTypeDefaultName = 'Computer')  ORDER BY vAlert.RaisedDateTime  10. Return to the Authoring console. In the Authoring console, in the Properties window for this report, on the Definition tab, click Load content from file, and then select the file for the report that you just updated. The content of the .rdl file appears in the text field.  11. You now must add a ReportParameters element to the report definition. In the Properties window for the report, click the Definition tab, and then click Edit in External Editor. The report opens in the XML editor of your choice. Search for a ReportParameters section in the file, and if it exists, delete it. Following the closing /DataSources tag in the report, add the following code:  <ReportParameters>  <ReportParameter Name="StartDate\_BaseType">  <DataType>String</DataType>  <Prompt>[Start Date Base]</Prompt>  </ReportParameter>  <ReportParameter Name="StartDate\_BaseValue">  <DataType>DateTime</DataType>  <Prompt>[From]</Prompt>  </ReportParameter>  <ReportParameter Name="StartDate\_OffsetType">  <DataType>String</DataType>  <Prompt>[Start Date Offset Type]</Prompt>  </ReportParameter>  <ReportParameter Name="StartDate\_OffsetValue">  <DataType>Integer</DataType>  <AllowBlank>true</AllowBlank>  <Prompt>[Start Date Offset]</Prompt>  </ReportParameter>  <ReportParameter Name="EndDate\_BaseType">  <DataType>String</DataType>  <Prompt>[End Date Base]</Prompt>  </ReportParameter>  <ReportParameter Name="EndDate\_BaseValue">  <DataType>DateTime</DataType>  <Prompt>[To]</Prompt>  </ReportParameter>  <ReportParameter Name="EndDate\_OffsetType">  <DataType>String</DataType>  <Prompt>[End Date Offset Type]</Prompt>  </ReportParameter>  <ReportParameter Name="EndDate\_OffsetValue">  <DataType>Integer</DataType>  <AllowBlank>true</AllowBlank>  <Prompt>[End Date Offset]</Prompt>  </ReportParameter>  <ReportParameter Name="TimeZone">  <DataType>String</DataType>  <Prompt>[Time Zone]</Prompt>  </ReportParameter>  <ReportParameter Name="TimeZoneName">  <DataType>String</DataType>  <Nullable>true</Nullable>  <AllowBlank>true</AllowBlank>  <Prompt>[Time Zone Name]</Prompt>  </ReportParameter>  </ReportParameters>  12. You must now add the query parameters to the DataSet for this report. Following the /CommandText tag that contains the query for the report, add the following code:  <QueryParameters>  <QueryParameter Name="@StartDate">  <Value>=Code.ToDbDate(Code.GetReportStartDate())</Value>  </QueryParameter>  <QueryParameter Name="@EndDate">  <Value>=Code.ToDbDate(Code.GetReportEndDate())</Value>  </QueryParameter>  </QueryParameters>  If you edited the query in BIDS, and the QueryParameter section already exists, you only have to replace the Value elements.  13. The smart parameter block requires additional code to process the report parameters. Locate the closing /DataSets tag, add the Code element, and then insert the code block from the previous section.  14. After the /Code tag, add the following code:  <CodeModules>  <CodeModule>Microsoft.EnterpriseManagement.Reporting.Code, Version=6.0.0.0, Culture=neutral</CodeModule>  </CodeModules>  This element is required to enable the code in the Code section to work correctly.  15. Save the .rdl file, and then close the external editor. The updated .rdl file appears in the RDL field on the Properties window. Delete the first line in that field, which looks like this:  <?xml version="1.0" encoding="utf-8"?>  Click OK to close the Properties window. Save the management pack.  16. Locate the file where your management pack is saved, and open it with the XML editor of your choice. Within the References section of the file, add the following code:  <Reference Alias="MicrosoftSystemCenterDataWarehouseReportLibrary">  <ID>Microsoft.SystemCenter.DataWarehouse.Report.Library</ID>  <Version>6.1.7221.0</Version>  <PublicKeyToken>31bf3856ad364e35</PublicKeyToken>  </Reference>  17. Save the report, and then deploy it as discussed in [Deploying Reports](#z18c5f1eb721f45209918d45203a3ef7f). |

Localizing Reports

For people who speak other languages than English, you can localize the report. There are no localization tools in Microsoft SQL Server Reporting Services, and therefore Business Intelligence Development Studio. However, Operations Manager 2007 R2 supports multiple languages by using language packs. Supporting another language requires defining report strings for text that you want to localize, and then providing the appropriate display strings in the language pack section of the management pack. By using the Authoring console, you can create report strings, but providing the content itself requires that you edit the .xml file. In addition, you must also use some code and references from the generic report library.

Activating Language Support in a Management Pack

To prepare the report for localization, you first must create report strings in the management pack. The Authoring console has a tool to assist you with that.

To add report strings to a management pack

|  |
| --- |
| 1. In the Authoring console, open the management pack that contains the report. Click the Reporting tab. Right-click the report that you want to modify, and select Properties.  2. Click the Report Strings tab. On the Report Strings page, click Create. The Choose a unique identifier dialog box opens. Enter an identifier for the report string. This identifier must be unique in the report and cannot be changed after it is created. You must preface the identifier with R., for example, R.ReportTitle, so that the code that you will add later functions correctly. Click OK when you are finished.  3. The string you just created appears in the Report Strings list. Select the string, and then click Edit. The Properties page for that string appears. In the Name box, enter the value for that string in the default language; that is, the text you want to appear when the string is used. In the Description box, you can optionally enter a description of what this string is used for. The report users do not see this text, but it can be helpful to add a description if you have many strings in your report. Click OK to close this Properties page.  4. Add any additional report strings that you need for this report, and set their Name properties.  5. Click OK to close the report properties. Save the management pack. |

You now have to add some language packs to your management pack. The Authoring console has a feature that lets you create new language packs and view what display strings and knowledge articles are used in existing language packs. However, you cannot create new display strings or edit existing ones with the Authoring console. For creating or editing display strings, you must edit the .xml file.

To add language packs to the management pack

|  |
| --- |
| 1. In the Authoring console, click the Language Packs tab. The left pane shows a language pack for United States English (ENU) and for your default language, if it is not ENU. A check mark indicates the default language pack.  2. In the left pane, right-click and select New Language Pack. The New Language Pack wizard opens, containing just one control, the Language box. Select the language for which you want to add a language pack. Click Finish to close the wizard.  3. The new language pack appears in the list in the left pane. However, it does not contain any display strings or knowledge articles. Save the management pack, and then close the Authoring console. |

The report string has been created and has a value, and the empty language packs have been created. To provide values for those new language packs, you must edit the XML code for the management pack. The last section of the management pack .xml file is the LanguagePacks section, which is where the display strings for different languages are located. There is a LanguagePack element for each language pack shown in the Authoring console. By default, there is just one LanguagePack section, for your default language. Within the LanguagePack element is a DisplayStrings element, which contains several DisplayString elements. Each DisplayString element represents one item of displayed text in the management pack. The ElementID attribute of the DisplayString element uniquely identifies the element and is used to identify that display string elsewhere in the management pack. Each DisplayString element also has two subelements: Name and Description, just as the ReportStrings elements you just created do. You must add DisplayString elements for each ReportString element you created, add the code from the generic report library to process the display strings, and some additional code to display the strings in the appropriate locations in your report.

To edit the XML code to include display strings

|  |
| --- |
| 1. Locate the .xml file for the management pack, and open it in the XML Editor of your choice.  2. Search for the term “ReportStrings” in this file. You will find a section of the file where the ReportStrings element is declared, which looks like the following code.  <ReportStrings>  <ReportString ID="R.ReportTitle" />  </ReportStrings>  The ReportString elements that you created in the previous procedure are found here. Notice the ReportString ID element and attribute with the value of R.ReportTitle in this example.  3. In the LanguagePacks section of the file, locate the corresponding DisplayString element, where the SubElementID attribute of the DisplayString element matches the ReportString ID attribute, as in the following code.  <DisplayString ElementID="AvailabilityReportingMP.AvailabilityReport" SubElementID="R.ReportTitle">  <Name>Availability Report</Name>  <Description>Indicates the title of the report.</Description>  </DisplayString>  You can see the text that you entered previously between the Name and Description elements.  4. You also find a LanguagePack element for each additional language pack that you created in the previous procedure with its associated 3-letter value, for example:  <LanguagePack ID="DEU" IsDefault="false" />  5. Delete the slash mark / of the closed element. Then add the following code to provide the necessary display strings for this LanguagePack element and a new closing tag.  <LanguagePack ID="DEU" IsDefault="false">  <DisplayStrings>  <DisplayString ElementID="" SubElementID="">  <Name></Name>  <Description></Description>  </DisplayString>  </DisplayStrings>  </LanguagePack>  6. If this language is the default language for this management pack, set the IsDefault attribute value to true. Only one language pack can be the default language, so ensure that the IsDefault attribute value on any other language packs is set to false.  7. In the DisplayString element, set the ElementID attribute to the ID of the report, which should be the same as in the DisplayString element for the default language. Set the SubElementID attribute to be the same as the ID attribute for the report string you created.  8. Between the Name tags, insert the text that should be displayed for this element when the selected language pack is used. Optionally, between the Description tags, enter a description of this element.  9. If your report already contains a ReportParameters section, locate it in the file. If you have not defined any ReportParameters elements, this section might not yet be present in your report; therefore you must add it. Following the closing /DataSources tag, add the following code.  <ReportParameters>  </ReportParameters>  10. Between the opening and closing ReportParameters tags, add the following code.  <ReportParameter Name="LT\_Report">  <DataType>String</DataType>  <DefaultValue>  <DataSetReference>  <DataSetName>LT\_Report\_Query</DataSetName>  <ValueField>ValueXml</ValueField>  </DataSetReference>  </DefaultValue>  <ValidValues>  <DataSetReference>  <DataSetName>LT\_Report\_Query</DataSetName>  <ValueField>ValueXml</ValueField>  <LabelField>ValueName</LabelField>  </DataSetReference>  </ValidValues>  <Hidden>true</Hidden>  </ReportParameter>  <ReportParameter Name="LT\_Table">  <DataType>String</DataType>  <DefaultValue>  <DataSetReference>  <DataSetName>LT\_Table\_Query</DataSetName>  <ValueField>ValueXml</ValueField>  </DataSetReference>  </DefaultValue>  <ValidValues>  <DataSetReference>  <DataSetName>LT\_Table\_Query</DataSetName>  <ValueField>ValueXml</ValueField>  <LabelField>ValueName</LabelField>  </DataSetReference>  </ValidValues>  <Hidden>true</Hidden>  </ReportParameter>  These ReportParameter elements are necessary for the correct functioning of the code that you will add in the next step.  11. The two ReportParameter elements you just added require their own datasets. Locate the DataSet element in the file. This element is already present, and includes the dataset for your existing report query. Following the last /DataSet end tag that currently exists, add the following code.  <DataSet Name="LT\_Report\_Query">  <Fields>  <Field Name="ValueXml">  <DataField>XML\_F52E2B61-18A1-11d1-B105-00805F49916B</DataField>  </Field>  <Field Name="ValueName">  <Value>="LT"</Value>  </Field>  </Fields>  <Query>  <DataSourceName>DataWarehouseMain</DataSourceName>  <CommandType>StoredProcedure</CommandType>  <CommandText>ReportDisplayStringGet</CommandText>  <QueryParameters>  <QueryParameter Name="@ManagementGroupGuid">  <Value>=Code.GetCallingManagementGroupId()</Value>  </QueryParameter>  <QueryParameter Name="@ManagementPackSystemName">  <Value>[Management Pack ID]</Value>  </QueryParameter>  <QueryParameter Name="@ReportSystemName">  <Value>[Report ID]</Value>  </QueryParameter>  <QueryParameter Name="@LanguageCode">  <Value>=Code.GetReportLocLanguageCode()</Value>  </QueryParameter>  <QueryParameter Name="@Scope">  <Value>R.%</Value>  </QueryParameter>  </QueryParameters>  </Query>  </DataSet>  <DataSet Name="LT\_Table\_Query">  <Fields>  <Field Name="ValueXml">  <DataField>XML\_F52E2B61-18A1-11d1-B105-00805F49916B</DataField>  </Field>  <Field Name="ValueName">  <Value>="LT"</Value>  </Field>  </Fields>  <Query>  <DataSourceName>DataWarehouseMain</DataSourceName>  <CommandType>StoredProcedure</CommandType>  <CommandText>ReportDisplayStringGet</CommandText>  <QueryParameters>  <QueryParameter Name="@ManagementGroupGuid">  <Value>=Code.GetCallingManagementGroupId()</Value>  </QueryParameter>  <QueryParameter Name="@ManagementPackSystemName">  <Value>[Management Pack ID]</Value>  </QueryParameter>  <QueryParameter Name="@ReportSystemName">  <Value>[Report ID]</Value>  </QueryParameter>  <QueryParameter Name="@LanguageCode">  <Value>=Code.GetReportLocLanguageCode()</Value>  </QueryParameter>  <QueryParameter Name="@Scope">  <Value>RT.%</Value>  </QueryParameter>  </QueryParameters>  </Query>  </DataSet>  Without this code, the localization process does not work. Both DataSet elements contain the element QueryParameter Name="@ManagementPackSystemName". Replace the content in the Value element in this QueryParameter element with the ID for the management pack. Also, both DataSet elements contain an element QueryParameter Name="@ReportSystemName". Replace the content in the Value element in this QueryParameter element with the ID for the report.  12. After the /Datasets end tag, add the following code.  <Code>  Const LocTableStringQuery As String = "R/S[@N='{0}']/text()"  Const TimeZoneParameterName As String = "TimeZone"  Const SD\_BaseTypeParameterName As String = "StartDate\_BaseType"  Const SD\_BaseValueParameterName As String = "StartDate\_BaseValue"  Const SD\_OffsetTypeParameterName As String = "StartDate\_OffsetType"  Const SD\_OffsetValueParameterName As String = "StartDate\_OffsetValue"  Const ED\_BaseTypeParameterName As String = "EndDate\_BaseType"  Const ED\_BaseValueParameterName As String = "EndDate\_BaseValue"  Const ED\_OffsetTypeParameterName As String = "EndDate\_OffsetType"  Const ED\_OffsetValueParameterName As String = "EndDate\_OffsetValue"  Const IsRelativeTimeSupported As Boolean = False  Const TimeTypeParameterName As String = "TimeType"  Const TimeWeekMapParameterName As String = "TimeWeekMap"  Dim LocTables As System.Collections.Generic.Dictionary(Of String, Microsoft.EnterpriseManagement.Reporting.XmlStringTable)  Dim ReportTimeZone As Microsoft.EnterpriseManagement.Reporting.TimeZoneCoreInformation  Dim ReportStartDate As DateTime  Dim ReportEndDate As DateTime  Dim ReportTime As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor.RelativeTime  Dim ReportCulture As System.Globalization.CultureInfo  Dim ParameterProcessor As Microsoft.EnterpriseManagement.Reporting.ParameterProcessor  Protected Overrides Sub OnInit()  LocTables = new System.Collections.Generic.Dictionary(Of String, Microsoft.EnterpriseManagement.Reporting.XmlStringTable)()  ReportTimeZone = Nothing  ReportStartDate = DateTime.MinValue  ReportEndDate = DateTime.MinValue  ReportTime = Nothing  ReportCulture = System.Globalization.CultureInfo.GetCultureInfo(Report.User("Language"))  ParameterProcessor = New Microsoft.EnterpriseManagement.Reporting.ParameterProcessor(ReportCulture)  End Sub  Public Function GetCallingManagementGroupId() As String  Return Microsoft.EnterpriseManagement.Reporting.ReportingConfiguration.ManagementGroupId  End Function  Public Function GetReportLocLanguageCode() As String  Return ReportCulture.ThreeLetterWindowsLanguageName  End Function  Public Function GetLocTable(Name As String) As Microsoft.EnterpriseManagement.Reporting.XmlStringTable  Dim LocTable As Microsoft.EnterpriseManagement.Reporting.XmlStringTable  If Not LocTables.TryGetValue(Name, LocTable) Then  LocTable = New Microsoft.EnterpriseManagement.Reporting.XmlStringTable(LocTableStringQuery, Report.Parameters(Name).Value)  LocTables.Add(Name, LocTable)  End If  Return LocTable  End Function  </Code>  13. After the /Code end tag, add the following code.  <CodeModules>  <CodeModule>Microsoft.EnterpriseManagement.Reporting.Code, Version=6.0.0.0, Culture=neutral</CodeModule>  </CodeModules>  This CodeModules element is required to access the code in the Code.  14. Next, you must indicate where you want your report string to be displayed. Locate the point inside the body tag where you want the localized text to appear. If you generated the report by using SQL Server 2008 Business Intelligence Development Studio (BIDS), this will most likely be in a TextRun element. Within that TextRun element, replace the text inside the Value element with code similar to the following code.  <Value>=Code.GetLocTable("LT\_Report").GetString("R.ReportTitle", "[Report Title]")</Value>  In place of R.ReportTitle, insert the ID of the ReportString element that you defined earlier. In place of [Report Title], insert the default text; that is, the text that will appear if no report string with that ID is found.  15. Repeat the previous step for all the report strings that you defined.  16. Save the file and redeploy the management pack. For more detailed information, see [Deploying Reports](#z18c5f1eb721f45209918d45203a3ef7f). |

After deploying the report, if the user opens the report by using an operating system set to another language, the report opens with the display strings configured for that language.

Warning

Management packs that include many display strings can become very large. A recommended practice to reduce the file size of the management packs is to move the language pack sections into separate management packs. This requires sealing the parent management pack.

Using Stored Procedures in a Custom Report

You can save a query in a Transact-SQL stored procedure to create a report. Stored procedures provide more efficiency and security over embedded SQL queries, but they must be created by a script. For more information about stored procedures, see [Stored Procedure How-To Topics](http://go.microsoft.com/fwlink/?LinkId=220266) (http://go.microsoft.com/fwlink/?LinkId=220266).

To create a stored procedure

|  |
| --- |
| 1. Open Microsoft SQL Server Management Studio and connect to the database server.  2. In Object Explorer, connect to an instance of the database engine, and then expand that instance.  3. Expand Databases, expand the OperationsManagerDW database, and then expand Programmability.  4. Right-click Stored Procedures, and select New Stored Procedure. A query editing pane opens.  5. This pane shows a script that creates the stored procedure in the database for later use. A skeleton of the script is provided for you; placeholder items that you must replace are indicated with brackets (<>).  6. Replace the placeholder text following CREATE PROCEDURE with the name that you want to give the procedure. Make a note of this name; you must have it when you design the report.  7. If you are using a specific user to execute the report, for example, the readonly user recommended in [Setting up the Environment](#z50474d3c3da644c480600afc8cb7863d), that user must have permission to use the stored procedure. Locate the following lines in the script:  USE [OperationsManagerDW]  GO  After those lines, add this code:  GRANT EXECUTE TO readonly  GO  This provides the readonly user who has permission to run the script.  8. If you are not using parameters in this query, you can comment out (preface with --) the two lines that begin <@Param. If you are using parameters, provide the name of the parameter, the data type, and the default value, if any. Note the names of the parameters for later use in designing your report. For more about how to use parameters with your query, see [Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b).  9. Replace the SELECT statement with the query that you have created to use in your report.  10. On the toolbar, click Parse to test the syntax of your query.  11. After the syntax is correct, click Execute to create the procedure in the database.  12. Close SQL Server Management Studio. |

After you have created the stored procedure, it is available to use instead of the query in any report you create.

To use a stored procedure in your report

|  |
| --- |
| 1. Create a new report or open an existing report. For more information about creating a report, see [Using a Report Builder](#zbbd29df9345b4b0697f6edc07f80ab47). If you are creating a new report, use the wizard until you reach the point of creating the query. If you are using an existing report, right-click the dataset for this report, and then select Query. The Query Designer pane opens.  2. In the Query Designer, click Edit As Text. The Query Designer switches to text editing mode.  3. Click the Command type selection box, and then select Stored Procedure. In the text box, type the name of the stored procedure that you created earlier. Click ! to test the stored procedure.  4. Click OK to close the Query Designer. |

The report now returns data as if you had used the Transact-SQL query directly.

The procedures discussed so far work with stored procedures that are created on the local report server. If you are using a stored procedure to be distributed with a management pack, you must add a script to the management pack to manage that stored procedure. The script requires three separate sections, for installing, uninstalling, and upgrading the stored procedure. The Authoring console has tools to assist with this.

Warning

Perform this procedure on existing reports only. If you create the scripts in the Authoring console at the same time as you create the report definition itself, you cannot save the scripts. Create the report definition, apply it to the management pack, save the management pack, and then create the scripts as described here.

To add a stored procedure to a management pack

|  |
| --- |
| 1. Open the management pack containing the report in the Authoring console. Click the Reporting tab.  2. Right-click the report in which you are using the stored procedure, and select Properties. The Properties page for that report opens.  3. Click the Dependencies tab. In the Data Warehouse Scripts section of this tab, click Create. The Choose a unique identifier dialog box opens. Enter an identifier that you can use to locate the script later. Click OK. The Properties page for this script opens.  4. On the General tab, in the Name section, enter a user-readable name that can be used to identify the script. Optionally, in the Description section, enter a description of what the script is used for.  5. The Properties page has a tab for each script that you must provide. Click the Install tab. In this box, enter the Transact-SQL code required to install your stored procedure. This code begins with the following lines:  IF NOT EXISTS (SELECT \* FROM sysobjects WHERE type = 'P' AND name = 'stored\_procedure\_name')  BEGIN  EXECUTE ('CREATE PROCEDURE dbo.[stored\_procedure\_name] AS RETURN 1')  END  GO  Replace stored\_procedure\_name with the name that you have selected for your stored procedure. After the line that shows GO, enter the Transact-SQL code for your stored procedure. If your stored procedure is already written and saved to a file, click Load content from file, navigate to the location of the file, and click OK. If you prefer to enter the code in an external editor, click Edit in external editor, and the editor of your choice opens.  6. Click the Uninstall tab. In this box, enter the Transact-SQL code required to install your stored procedure. This code is usually much simpler than the installation code, as it only requires a DROP PROCEDURE statement. That code should look like this:  IF EXISTS (SELECT \* FROM sysobjects WHERE type = 'P' AND name = 'stored\_procedure\_name')  BEGIN  DROP PROCEDURE dbo.[stored\_procedure\_name]  END  GO  Again, replace stored\_procedure\_name with the name that you have selected for your stored procedure. If your stored procedure is already written and saved to a file, click Load content from file, navigate to the location of the file, and click OK. If you prefer to enter the code in an external editor, click Edit in external editor, and the editor of your choice opens.  7. Click the Upgrade tab. In this box, enter the Transact-SQL code required to upgrade your stored procedure. This code is almost the same as the code for the installation procedure. However, the IF NOT EXISTS clause is not required, because the upgrade will not run if the procedure does not already exist. Instead, begin with this code, which should already be present in your stored procedure:  ALTER PROCEDURE [dbo].[stored\_procedure\_name]  Again, replace stored\_procedure\_name with the name that you have selected for your stored procedure. Then add the code that defines the stored procedure. Ensure that the check box Automatically upgrade previous versions of the script is selected. If your stored procedure is already written and saved to a file, click Load content from file, navigate to the location of the file, and then click OK. If you prefer to enter the code in an external editor, click Edit in external editor, and the editor of your choice opens.  Click OK to close the script properties. The script now appears in the Data Warehouse Scripts section of the report properties. Click OK to close this Properties page.  8. Save the management pack, but do not deploy it yet.  9. You must perform an additional step that the Authoring console does not do for you. You must edit the XML directly. Open the .xml file for the management pack in the XML editor of your choice. The scripts you just created are located in the DataWarehouseScripts section of the management pack. If you examine this section of the management pack, you will see separate sections with the tags <Install>, <Uninstall>, and <Upgrade>. Immediately following the Install tag is an opening <CDATA> tag, which encloses the content of the script. The Upgrade section also has a <CDATA> tag. The Uninstall section does not have a <CDATA> tag; you must add it manually. Add the opening and closing <CDATA> tags so that the Uninstall section now looks like this:  <Uninstall><![CDATA[IF EXISTS (SELECT \* FROM sysobjects WHERE type = 'P' AND name = 'stored\_procedure\_name')  BEGIN  DROP PROCEDURE dbo.[stored\_procedure\_name]  END  GO]]></Uninstall>  10. Save the file, and then close the XML editor. Deploy the management pack. When the management pack is imported, the script installs the stored procedure on the local report server. |

Deploying Reports

After you have created your custom report, you must deploy it so that other users can run the report from the Reporting tab of the Operations console. There are three ways to deploy a custom report that was created in SQL Server Business Intelligence Development Studio (BIDS):

 Add the report to a management pack and import it into Operations Manager. This is the recommended method because it allows the most flexible deployment. If you use any custom controls defined in the .rpdl file, as described in [Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b), you must deploy the report to a management pack in order to use them.

 Deploy the report directly from BIDS to the SQL Server Reporting Services (SSRS).

 Save the report as an .rdl file and import it into SSRS.

All three options are covered in this document.

Deploying the Report to a Management Pack

Adding the report to a management pack is a lengthier process than the other two deployment methods. It provides the most flexibility for your report because it lets you modify the XML of the report directly, and enables use of the smart parameter block. Also, if the reporting server in your management group is reinstalled, the Operations console will be unable to access reports deployed to the server. Adding the report to a management pack eliminates that problem because the report management server will automatically deploy the report to any new reporting server. If you do not have an existing management pack to add the report to, you must create one before you can add the report.

Note

To create or modify a management pack, you must download and install the Operations Manager Authoring Console. This is part of the [System Center Operations Manager Resource Kit](http://go.microsoft.com/fwlink/?LinkID=192098)(http://go.microsoft.com/fwlink/?LinkID=192098).

To create a new management pack

|  |
| --- |
| 1. Open the Authoring console.  2. In the Actions area on the right, select Create a new management pack. The New Management Pack wizard opens.  3. On the Management Pack Template page, select Empty Management Pack. In the Management Pack Identity field, provide a name for the management pack file. Click Next.  4. On the Name and Description page, in the Display Name field, provide the name for this management pack that will appear in the Operations console. The Description field is optional. However, we recommend that you provide a clear description of the management pack’s purpose.  5. Click Create. The new management pack is created. The name for the management pack appears in the title bar of the Authoring console.  6. Save the management pack by selecting File, and then Save. The Save Management Pack dialog box appears. The file name provided matches the one that you entered in the Management Pack Identity field. |

After you have created the management pack, you can add a report to it.

To add a report to a management pack

|  |
| --- |
| 1. In the Authoring console, click the Reporting tab. The Reporting page opens.  2. In the Reporting pane, click the Reports item. In the Actions pane, click New, and then New Report.  3. The Choose a unique identifier dialog box appears. Provide an identifier for this report. This name will only be used internally, and will not appear in the Operations console. Click OK.  4. The Properties page for the report opens. On the General tab, in the Id field, the identifier you just entered appears. In the Name field, enter a name for the report. This will be displayed in the Operations console. In the Description field, you may enter a description to help users understand the purpose of the report. In the Target field, if a target is specified, the report will be displayed in the Action pane of the operations console when an object of the target class is selected.  5. Click the Parameter Block tab. If your report uses parameters, in the Parameter Block field, copy the XML for the parameter block for this report, as defined in [Custom Report Parameters](#zf90f681aba3140cfb3437a2c32e0752b). If you have defined the parameter block in an .rpdl file, click the Load content from file button and browse to that file location. If you have not yet created the parameter block, click the Edit in external editor button, and the external editor will start. If you have no external XML editor defined, click Tools, and then Options. The Options page for the Authoring console opens. Click the Browse button in the Custom Editors area to select the external XML editor.  6. Click the Definition tab. Copy the XML for the report into the RDL field on this page. If you defined the report in another editor, such as BIDS, click the Load content from file button and browse to the location of the .rdl file. If you have not yet created the report definition, click the Edit in external editor button, and the external editor will start. If you have no external XML editor defined, click Tools, and then Options. The Options page for the Authoring console opens. Click the Browse button in the Custom Editors area to select the external XML editor.  7. In the RDL field, you can now modify the XML for the report. If you loaded the XML from a file, make sure that you delete the first line of XML. This line starts with <?xml version=.  8. On the Options tab, in the Visible section, select the True option button. If you do not change this option, the report will not be available from the Operations console.  9. Click OK. The report appears in the Reports pane.  10. Click Tools, and then click Export MP to Management Group. The Export MP to Management Group dialog box appears. Select the server that contains the management group you want to export to, and then click Connect. The management pack is automatically exported to the management group. |

If you deployed the management pack to a management group, the new management pack and its reports automatically appear in the Operations console. If you did not deploy the management pack to a management group, you must import the management pack into the Operations console.

To import the management pack from a file

|  |
| --- |
| 1. In the operations console, select the Administration tab. In the Administration pane, select the Management Packs item. In the Actions pane, click Import Management Packs. The Select Management Packs window appears.  2. Click the Add button and select Add from disk. The Online Catalog Connection dialog box appears. If the management pack you are importing has outside dependencies that are not stored locally, click Yes to search the online catalog for those dependencies. If the management pack has no dependencies, or if the dependencies are stored locally, click No.  3. The Select Management Packs to import window opens; browse to where the XML file for the management pack is saved, and then click Open. The Select Management Packs window now displays the management pack to import. If there is a green check mark next to the management pack name, the management pack is ready to import. If there is a red X next to the management pack name, there are errors in the management pack that prevent it from being imported. The Status details field will have information about how to fix these errors.  4. When all issues have been resolved, click Install. A progress bar will display the status of the installation. When installation is complete, click Close. |

You may have to wait from a few minutes to an hour for the management pack to propagate to the Operations console. The management pack now appears in the list of management packs in the Administration tab. If you select the Reporting tab, the management pack appears there also. Select the new management pack, and the report appears in the center pane.

Deploy the Report Directly from BIDS to SSRS

If your BIDS instance has direct access to the reporting server, deploying a report directly from BIDS to SSRS is relatively easy. However, deploying the report in this manner will not enable you to take advantage of the features available when you include the report in a management pack, such as using the smart parameter block.

To deploy a report directly from BIDS

|  |
| --- |
| 1. Open the report project in BIDS.  2. Verify that the report project has the correct URL for the report server. In Solution Explorer, right-click the project name, and then select Properties. The property pages window for this project opens, with the General tab selected.  3. In the Deployment section of this page, make sure that the TargetServerURL field is set to the correct URL for the report server. By default, this is http://localhost/ReportServer. If the URL is incorrect, you can change it here. Close the property pages window.  4. Click Build, and then click Deploy <project name>. The Output window at the bottom of the BIDS interface shows the status of the project deployment.  5. Open the Operations console, and select the Reporting tab. Expand the Reporting node if you have to. Your report project appears. You can now run this report from the Operations console, as you would any other report. |

If the report displays an error message that the data source has incorrect credentials, you must overwrite the data source.

To overwrite the data source

|  |
| --- |
| 1. In BIDS, in Solution Explorer, right-click the project, and then select Properties. The property pages window for this project opens, with the General tab selected.  2. In the Deployment section of this page, set the OverwriteDataSources property to true.  Click OK to close the property pages window.  3. Click Build, and then click Deploy <project name> to redeploy the project. |

The report should now be available on the Reporting tab of the operator console.

Save the Report as an .rdl File

If your installation of BIDS does not have direct access to the reporting server, you can save the report as an .rdl file and then import it into SSRS. Again, this deployment method will not let you take advantage of the features that come with including the report in a management pack, such as using the smart parameter block.

To deploy the report as an .rdl file

|  |
| --- |
| 1. Open the report project in BIDS.  Click File, and then click Save <report name>.rdl As. The Save As window appears. Save the .rdl file to a location that can be accessed by SSRS. You may have to contact a network administrator to determine the appropriate location.  2. Open the SSRS URL in a browser. By default, this is http://localhost/reports, which differs from the URL used for the TargetServerURL property in the report project. Click New Folder to create a new folder to contain the report. Give the new folder a name, and then click OK.  3. Click the folder to open it. Click Upload File in the menu bar for the folder. The Upload File window appears. Click Browse and browse to the location where you saved the .rdl file. Select the Overwrite item if it exists check box so that the report will be overwritten by a newer version. Click OK.  4. Open the Operations console, select the Reporting tab. Expand the Reporting node if you have to. Your report project appears. You can now run this report from the console, as you would any other report. |

Scheduling and Sending Reports

The procedures in this section enable you to create a schedule to run reports, send reports by e-mail, save reports to a file share, or store reports in the SQL Report Server cache.

Creating a Report Schedule

You can use the following procedure to create a schedule to run a report. However, before you do so, make sure that you create an Availability report and save it as a favorite.

To schedule a report to run

|  |
| --- |
| 1. In the Operations console, click Reporting.  2. In the Reporting pane, click Favorite Reports.  3. In the Favorite Reports pane, right-click the Availability report, and then click Schedule.  4. In the Subscribe to a Report Wizard, on the Delivery Settings page, do the following:  a. Type a description in the Description text box.  b. In the Delivery method list, click Report Server File Share.  c. Type a file name for the report in the File name text box.  d. Type a file path for the report in the Path text box.  Note  Because report scheduling supports the Universal Naming Convention (UNC), file names must not end in a backslash.  e. In the Render Format list, click the file format that you want for the report.  f. Type a user name in the User name text box, and then type a password in the Password text box.  g. Click the down arrow in the Write mode list, select the Write mode you want for subsequent files, and then click Next.  Note  The credentials that you entered in step 4g must have the same Write user rights on the file share that you entered in step 4d.  5. In the Subscribe to a Report Wizard, on the Subscription Schedule page, do the following:  a. Select one of the Generate the report options.  b. In The Subscription is effective beginning list, type a start date and a start time for the reports that you want to generate. You can also enter the date when this subscription will end in The subscription expires on list, and then click Next.  6. In the Subscribe to a Report wizard, on the Parameters page, specify a span of time for the report in the From and To lists.  7. Make any other changes you need for this report, and then click Finish. |

Sending a Report Through E-mail

You can configure Reporting to automatically send a published report from Operations Manager to another person through e-mail.

Before you can schedule a report for e-mail delivery, you must configure the e-mail settings in the Report Server by using the Reporting Server Configuration Manager.

To configure e-mail settings in the SQL Server 2005 Report Server

|  |
| --- |
| 1. Open the Reporting Services Configuration by clicking Start on the Windows taskbar, point to Programs, point to Microsoft SQL Server 2005, point to Configuration Tools, and then click Reporting Services Configuration.  2. In Reporting Services Configuration Manager, in the Report Server Installation Instance dialog box, click Connect.  3. In the navigation pane, click Email Settings.  4. In the E-mail Settings pane, enter the following:   An e-mail address that will be used as the sender address in the Sender Address text box.   The address of the SMTP server that will be used to send the e-mail messages in the SMTP Server text box.  5. Click Apply, and then click Exit. |

To e-mail scheduled reports

|  |
| --- |
| 1. In the Operations console, click the Reporting button.  2. In the Reporting pane, click Favorite Reports Reports.  3. In the Favorite Reports pane, right-click the Availability report, and then click Schedule.  4. In the Subscribe to a Report Wizard, on the Delivery Settings page, do the following:   Type a description in the Description text box.   In the Delivery method list, click Report Server E-Mail.   Type an e-mail address of the destination inbox to receive reports in the To text box. You can also type e-mail addresses in the Cc, Bcc, and the Reply To text boxes.   In the Render Format list, click the file format you want for the report.   In the Priority list, select the appropriate priority.   Type a subject for the e-mail in the Subject text box.   Click Next.  5. On the Subscription Schedule page, do the following:   Select one of the Generate the report options.   In the The Subscription is effective beginning list, type a start date and a start time for the reports that you want to generate. You can also enter the date when this subscription will end in The subscription expires on list, and then click Next.  6. On the Parameters page, specify a span of time for the report in the From and To lists, make any other changes that you need for this report, and then click Finish. |

Delivering a report to the SQL Server Report Server Cache

You can create a schedule for sending reports to the cache in the SQL Server 2005 Report Server. When you do this, you shorten the time required to retrieve a report, which can be helpful if a report is large or is accessed frequently. For more information about report caching, see <http://go.microsoft.com/fwlink/?LinkId=77536>.

To schedule the delivery of a report to the SQL Report Server Cache

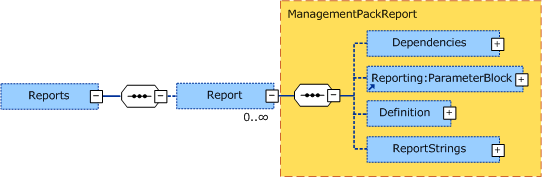
|  |
| --- |
| 1. In the Operations console, click Reporting.  2. In the Reporting pane, click Favorite Reports.  3. In the Favorite Reports pane, right-click the availability report you saved as a favorite, and then click Schedule.  4. In the Subscribe to a Report Wizard, on the Delivery Settings page, do the following:  a. Type a description in the Description text box.  b. In the Delivery method list, click Null Delivery Provider.  c. Click Next.  5. On the Subscription Schedule page, do the following:  a. Select one of the Generate the report options.  b. In the The Subscription is effective beginning list, type a start date and a start time for the reports that you want to generate. You can also enter the date when this subscription will end in The subscription will end list, and then click Next.  6. On the Parameters page, specify a span of time for the report in the From and To lists, make any other changes you need for this report, and then click Finish. |

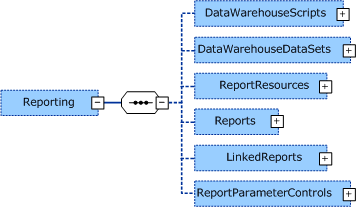
Appendix A - Reporting Management Pack Schema

Within a management pack such as the Microsoft.SystemCenter.DataWarehouse.Report.Library.xml, the Reporting section contains the necessary information for deploying related items such as the reports themselves, linked reports, and report parameter controls. When creating a reporting solution, it is important to understand the management pack schema. The full management pack schema is available at <http://go.microsoft.com/fwlink/?LinkId=111035>.

The following sections describe the basic elements of the schema.

Reports section:





Dependencies: Details any dependencies such as stored procedures required for the report to function.

Parameter Block: Defines the look of the Smart Parameter Header (SPH) that shows up in the console when the report is opened.

Each control in the SPH is bound to one or more ReportParameters

Also, some controls have user-definable properties, such as the LinkedPerformanceChartObjectPicker. All parameter block controls are defined in Appendix B.

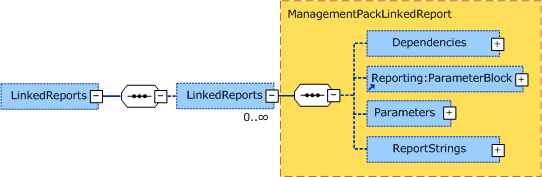
Definition: Contains the report RDL information. When you have an RDL file ready, the XML is copied directly into this section.

Note

Be sure to omit the xml declaration before pasting it in the definition section: <?xml version="1.0" encoding="utf-8"?>

Report Strings: Enables localization for the report.

Linked reports section:



Dependencies: Details any dependencies such as stored procedures required for the report to function.

Parameter Block: Defines the look of the Smart Parameter Header that shows up in the console when the report is opened.

Each Control in the SPH is bound to one or more ReportParameters

Also, some controls have user-definable properties, such as the LinkedPerformanceChartObjectPicker. All parameter block controls are defined in Appendix B.

Parameters: Passes pre-defined parameters to the report to enable the scoping of a generic report to user-defined values.

Report Strings: Enables localization for the report.

Report Parameter Controls section:

The Report parameters block is the place where all these parameters are entered. The Report parameters block consists of a set of report parameter controls, each of which defines a visual representation for one or more underlying report parameters.

Report parameter controls used for a particular report are defined in the RPDL file.

Here is the schema for them:

<xs:element name="Control" minOccurs="1" maxOccurs="unbounded">  
    <xs:complexType>  
        <xs:sequence>

            <xs:element name="ReportParameters" minOccurs="0" maxOccurs="1">  
                <xs:complexType>  
                    <xs:sequence>  
                        <xs:element name="ReportParameter"   
                                           minOccurs="1" maxOccurs="unbounded">  
                            <xs:complexType>  
                                <xs:sequence>  
                                    <xs:element name="Prompt" type="xs:string"  
                                                       minOccurs="0" maxOccurs="1" />  
                                </xs:sequence>  
                                <xs:attribute name="name" type="xs:token" use="required" />  
                                <xs:attribute name="binding" type="xs:token" use="optional" />  
                            </xs:complexType>  
                        </xs:element>  
                    </xs:sequence>  
                </xs:complexType>  
            </xs:element>

            <xs:element name="Properties" minOccurs="0" maxOccurs="1">  
                <xs:complexType>  
                    <xs:sequence>  
                        <xs:element name="Property"  
                                           minOccurs="1" maxOccurs="unbounded">  
                            <xs:complexType>  
                                <xs:sequence>  
                                    <xs:element name="Value" type="xs:string"  
                                                       minOccurs="1" maxOccurs="1" />  
                                </xs:sequence>  
                                <xs:attribute name="name" type="xs:token" use="required" />  
                            </xs:complexType>  
                        </xs:element>  
                    </xs:sequence>  
                </xs:complexType>  
            </xs:element>

        </xs:sequence>  
        <xs:attribute name="type" type="xs:token" use="optional" />  
        <xs:attribute name="columnSpan" type="xs:unsignedByte" use="optional" />  
        <xs:attribute name="rowSpan" type="xs:unsignedByte" use="optional" />  
    </xs:complexType>  
</xs:element>

As you can see from the schema, the parameter control has a list of underlying report parameters it uses. The way it uses these parameters is defined by the binding attribute. Binding is defined by specific control code. You cannot change the way it is handled, but you can set which report parameter is used. "Empty" or default binding is usually used for a report parameter for which the control sets value, but this is not a strict rule.

The next section is the parameter control properties. It is a name multi-value collection. These properties changes control behavior or visual appearance, for example controlling the maximum width or the background color.

All "out of the box" report parameter controls are defined in the Microsoft.SystemCenter.DataWarehouse.Report.Library management pack. The most commonly used controls are defined in Appendix B.

Appendix B - Parameter Values and Parameter Controls

This appendix provides information about specific parameter values and parameter controls that can be used when creating linked reports.

Parameter Values

DateTime

Parameters: StartDate\_BaseType,EndDate\_BaseType

BaseType Values:

 Fixed

 Today

 Monday

 Tuesday

 Wednesday

 Thursday

 Friday

 Saturday

 Sunday

 FirstDayMonth

 LastDayMonth

 FirstDayQuarter

 LastDayQuarter

 FirstDayYear

 LastDayYear

Parameters: StartDate\_OffsetType, EndDate\_OffsetType

OffsetType Values:

 None

 Day

 Week

 Month

 Quarter

 Year

Example:

<Parameter Name="StartDate\_BaseType">

<Value>Friday</Value>

</Parameter>

<Parameter Name="StartDate\_OffsetType">

<Value>Week</Value>

</Parameter>

<Parameter Name="StartDate\_OffsetValue">

<Value>-1</Value>

</Parameter>

<Parameter Name="EndDate\_BaseType">

<Value>Today</Value>

</Parameter>

<Parameter Name="EndDate\_OffsetType">

<Value>None</Value>

</Parameter>

<Parameter Name="EndDate\_OffsetValue">

<Value>0</Value>

</Parameter>

Aggregation Parameter

|  |  |
| --- | --- |
| Daily | 1 |
| Hourly | 0 |

Downtime Parameter

|  |  |
| --- | --- |
| Warning | 2 |
| Planned Maintenance | 6 |
| Unplanned Maintenance | 5 |
| Monitor disabled | 4 |
| Unmonitored | 3 |

Threshold

|  |  |
| --- | --- |
| 1 second(s) | 1 |
| 2 second(s) | 2 |
| 5 second(s) | 5 |
| 10 second(s) | 10 |
| 15 second(s) | 15 |
| 30 second(s) | 30 |
| 1 minute(s) | 60 |
| 2 minute(s) | 120 |
| 5 minute(s) | 300 |
| 10 minute(s) | 600 |
| 15 minute(s) | 900 |
| 30 minute(s) | 1800 |

Chart Scale

|  |  |
| --- | --- |
| Hourly | 0 |
| Daily | 1 |
| Monthly | 2 |

Severity

|  |  |
| --- | --- |
| Information | 0 |
| Warning | 1 |
| Critical | 2 |

Priority

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |

Criteria

|  |  |
| --- | --- |
| Defined In | 0 |
| Applied To | 1 |

Exclude Sealed MP

Boolean

Histogram

|  |  |
| --- | --- |
| No | 0 |
| Daily | 1 |
| Weekly | 2 |
| Monthly | 3 |

3D Chart

Boolean

Objects

Example:

<Data>  
 <Values Title="Test">  
 <Value>  
 <Object Use="Containment">9</Object>  
 <Rule>7c512fe8-a3c4-1049-9891-88b259cb5f47</Rule>

<Color>63,63,255</Color>

<Type>Line</Type>

<Scale>1</Scale>

</Value>

<Value>

<Object Use="Self">9</Object>

<Rule>406fc269-8b85-b13a-9079-86dd1df5831b</Rule>

<Color>0,159,0</Color>

<Type>Line</Type>

<Scale>1</Scale>

</Value>

</Values>

</Data>

Where

<Object>–Object Row Id

<Rule>–Rule GUID

<Color>–In the format of R,G,B.

<Type>–One of: Area, Column, Line, Point, Spline, SplineArea, or StepLine.

<Scale>–Double

Report Parameter Controls

The most commonly used report parameter controls in custom reports are as follows:

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.CheckedListBox

Description: A way to select multiple values from a list of valid ones. Replaces standard SSRS multi-value value picker.

Properties:

Bindings

Default (report parameter has to be a multi-value parameter with a list of valid values)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.CheckedListBox" rowSpan="1" columnSpan="1">

<ReportParameters>

<ReportParameter name="Severity"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

<Parameters>

<Parameter Name="Severity">

<Value>0,1,2</Value>

</Parameter>

</Parameters>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.ComboBox

Description: A way to select a value from a set. Replaces standard SSRS value picker control.

Properties:

Bindings

Default (report parameter has to have a list of valid values)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.ComboBox" rowSpan="1" columnSpan="1">

<ReportParameters>

<ReportParameter name="AggregationType"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

<Parameters>

<Parameter Name="AggregationType">

<Value>3,4</Value>

</Parameter>

</Parameters>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.BooleanPicker

Description: A way to enter parameter value of Boolean type. Replaces standard SSRS Boolean picker

Properties:

Bindings

Default (report parameter has to be Boolean type)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.BooleanPicker" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="Enable3D"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.DatePicker

Description: A way to select date value. Replaces standard SSRS date picker control.

Properties:

Bindings

Default (report parameter has to be DateTime type)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.DatePicker" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />

<ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue"> <Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>

</ReportParameter>

<ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />

<ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />

<ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />

<ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>

</ReportParameter>

<ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />

<ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.DateTimePicker

Description: A way to select date and time value.

Properties:

Bindings

Default (report parameter has to be DateTime type)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.DateTimePicker" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="TimeZone" binding="TimeZone" />

<ReportParameter name="TimeZoneName" binding="TimeZoneName" />

<ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />

<ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>

</ReportParameter>

<ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />

<ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />

<ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />

<ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>

</ReportParameter>

<ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />

<ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.NumericUpDown

Description: A way to select an integer value.

Properties:

Bindings

Default (report parameter has to be Integer type)

Parameters

Minimum (minimum possible integer value)

Maximum (maximum possible integer value)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.DateTimePicker" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="TimeZone" binding="TimeZone" />

<ReportParameter name="TimeZoneName" binding="TimeZoneName" />

<ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />

<ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>

</ReportParameter>

<ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />

<ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />

<ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />

<ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>

</ReportParameter>

<ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />

<ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.TextBox

Description: A way to enter string parameter value. Replaces standard SSRS text box control.

Properties:

Bindings

Default

Parameters

Multiline (True/False)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.NumericUpDown" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="EventID"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

<Parameters>

<Parameter Name="EventID">

<Value>26319</Value>

</Parameter>

</Parameters>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.RelativeDateTimePicker

Description: A way to select relative date time range. This class is used on the report side to process values entered by this control.

Properties:

Bindings

TimeZone (time zone reference)

TimeZoneName (name of the time zone displayed in the report)

StartDate\_BaseType (start date base type)

StartDate\_BaseValue (start date base value)

StartDate\_OffsetType (type of start date offset)

StartDate\_OffsetValue (start date offset value)

EndDate\_BaseType (end date base type)

EndDate\_BaseValue (end date base value)

EndDate\_OffsetType (type of end date offset)

EndDate\_OffsetValue (end date offset value)

Parameters

MaxWidth (control maximum width in pixels)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.RelativeDateTimePicker" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="TimeZone" binding="TimeZone" />

<ReportParameter name="TimeZoneName" binding="TimeZoneName" />

<ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />

<ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>

</ReportParameter>

<ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />

<ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />

<ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />

<ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>

</ReportParameter>

<ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />

<ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.BusinessRelativeDateTimePicker

Description: A way to select relative date time range with business hours. This class is used on the report side to process values entered by this control.

Properties:

Bindings

TimeZone (time zone reference)

TimeZoneName (name of the time zone displayed in the report)

StartDate\_BaseType (start date base type)

StartDate\_BaseValue (start date base value)

StartDate\_OffsetType (type of start date offset)

StartDate\_OffsetValue (start date offset value)

EndDate\_BaseType (end date base type)

EndDate\_BaseValue (end date base value)

EndDate\_OffsetType (type of end date offset)

EndDate\_OffsetValue (end date offset value)

TimeType (type of time entered (business or regular))

TimeWeekMap (days of week selected for business time)

Parameters

MaxWidth (control maximum width in pixels)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.BusinessRelativeDateTimePicker" rowSpan="2" columnSpan="2">

<ReportParameters>

<ReportParameter name="TimeZone" binding="TimeZone" />

<ReportParameter name="TimeZoneName" binding="TimeZoneName" />

<ReportParameter name="StartDate\_BaseType" binding="StartDate\_BaseType" />

<ReportParameter name="StartDate\_BaseValue" binding="StartDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.StartDateTime</Prompt>

</ReportParameter>

<ReportParameter name="StartDate\_OffsetType" binding="StartDate\_OffsetType" />

<ReportParameter name="StartDate\_OffsetValue" binding="StartDate\_OffsetValue" />

<ReportParameter name="EndDate\_BaseType" binding="EndDate\_BaseType" />

<ReportParameter name="EndDate\_BaseValue" binding="EndDate\_BaseValue">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.EndDateTime</Prompt>

</ReportParameter>

<ReportParameter name="EndDate\_OffsetType" binding="EndDate\_OffsetType" />

<ReportParameter name="EndDate\_OffsetValue" binding="EndDate\_OffsetValue" />

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.MonitoringObjectPicker

Description: A way to select monitoring object for report. List of Monitoring Object IDs are passed to the report as a result value (default binding).

Properties:

Bindings

Default (report parameter has to be multi-value Integer type)

GroupList (list of management groups objects are allowed to be selected from)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.MonitoringObjectPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="MonitoringObjectId" />

<ReportParameter name="ManagementGroupId" binding="GroupList" />

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.MonitoringObjectXmlPicker

Description: For object Selection input parameters as XML

Properties:

Bindings

Default (report parameter has to be multi-value Integer type)

GroupList (list of management groups objects are allowed to be selected from)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.MonitoringObjectXmlPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.ObjectList</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceObjectPicker

Description: The Performance counter selection is based on the Object Selector. A performance counter selection has to support several objectives:

a) Scale is a parameter needed to allow comparing two counters with different value ranges.

b) The Dundas chart control allows changing of the chart type per series and also changing of the z-order (which series is sorted to the front, which to the back)

Properties:

Bindings

Default (report parameter has to be multi-value Integer type)

GroupList (list of management groups objects are allowed to be selected from)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceObjectPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.ObjectList</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceChartObjectPicker

Description: For performance Chart Object Selection

Properties:

Bindings

Default (report parameter has to be multi-value Integer type)

GroupList (list of management groups objects are allowed to be selected from)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceChartObjectPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="ObjectList"/>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.ObjectList</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceRulePicker

Description: For Performance Rules Selection

Properties:

Bindings

Default (report parameter has to be multi-value Integer type)

GroupList (list of management groups objects are allowed to be selected from)

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceRulePicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="ObjectList"/>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.PerformanceRule</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceRuleInstancePicker

Description: For Performance Rule Instances Selection

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.PerformanceRuleInstancePicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="ObjectList"/>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.PerformanceRule</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.LinkedPerformanceObjectPicker

Description: For Performance Object Selection

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.LinkedPerformanceObjectPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="ObjectList"/>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.PerformanceRule</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.LinkedPerformanceChartObjectPicker

Description: For performance Chart Object selection.

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.LinkedPerformanceChartObjectPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="ObjectList"/>

<ReportParameter name="MonitoringObjectId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.PerformanceRule</Prompt>

</ReportParameter>

<ReportParameter name="ManagementGroupId" binding="GroupList"/>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.ReportColumnPicker

Description: For Report Column Selection

Sample:

<LinkedReport>

<ParameterBlock>

<Controls>

<Control type="Microsoft.SystemCenter.DataWarehouse.Report.ParameterControl.ReportColumnPicker" rowSpan="2" columnSpan="3">

<ReportParameters>

<ReportParameter name="ColumnId">

<Prompt>Microsoft.SystemCenter.DataWarehouse.Report.Library!Microsoft.SystemCenter.DataWarehouse.Report.ParameterPrompt.ColumnList</Prompt>

</ReportParameter>

</ReportParameters>

</Control>

</Controls>

</ParameterBlock>

</LinkedReport>

Appendix C - Data Types and Sample Queries

This appendix provides information about specific data types and sample queries for operational, performance, and discovery data that can be used when creating custom reports.

For State, Alert, and Performance data, you need to know which managed entity you want to query. To find this information, you get the ManagedEntityRowID from the vManagedEntity view. For groups, you can use the following query to return all managed entities within the group:

1. First, retrieve the ManagedEntityRowID for the group:  
select \* from vmanagedentity  
where fullname like 'Microsoft.SystemCenter.AllComputersGroup'

2. Note the ManagedEntityRowID.

3. Run the following query, with these changes:  
- Update the start(1st)and end(2nd) dates to return all entities that were a member of that group during these dates  
- Replace the Containment Value below (166) with the ManagedEntityRowID returned in Step 1.

EXEC [dbo].[Microsoft\_SystemCenter\_DataWarehouse\_Report\_Library\_ReportObjectLis tParse]'2008-01-01 10:50:23.150','2008-02-05 10:50:23.150', '<Data><Objects><Object Use="Containment">166</Object></Objects></Data>

4. You can join the resulting list of managed entities to the performance, state, and alert tables to retrieve relevant information.

Operational Data

State

State information is derived from different types of monitoring and is stored directly in the OperationsManagerDW database. State is summarized per object and per time, and it is made available in different time units: hourly, sub-hourly, and raw.

vStateHourlyFull

|  |  |
| --- | --- |
| Column | Datatype |
| DateTime | datetime |
| Date | datetime |
| Time | char(8) |
| ManagedEntityMonitorRowId | int |
| ManagedEntityRowID | int |
| IntervalDurationMilliseconds | int |
| InYellowStateMilliseconds | int |
| InRedStateMilliseconds | int |
| InPlannedMaintenanceMilliseconds | int |
| InUnplannedMaintenanceMilliseconds | int |
| InDisabledStateMilliseconds | int |
| HealthServiceUnavailableMilliseconds | int |
| InWhiteStateMilliseconds | int |
| InGreenStateMilliseconds | int |

vStateDailyFull

|  |  |
| --- | --- |
| Column | Datatype |
| DateTime | datetime |
| Date | datetime |
| Time | char(8) |
| ManagedEntityMonitorRowId | int |
| ManagedEntityRowID | int |
| IntervalDurationMilliseconds | int |
| InYellowStateMilliseconds | int |
| InRedStateMilliseconds | int |
| InPlannedMaintenanceMilliseconds | int |
| InUnplannedMaintenanceMilliseconds | int |
| InDisabledStateMilliseconds | int |
| HealthServiceUnavailableMilleseconds | int |
| InWhiteStateMilliseconds | int |
| InGreenStateMilliseconds | int |

vStateRaw

|  |  |
| --- | --- |
| Column | Datatype |
| EventOriginId | uniqueidentifier |
| ManagedEntityMonitorRowId | int |
| DateTime | datetime |
| OldHealthState | tinyint |
| NewHealthState | tinyint |

Query:

SELECT

vStateHourlyFull.DateTime,

vStateHourlyFull.InRedStateMilliseconds,

vStateHourlyFull.InYellowStateMilliseconds,

vStateHourlyFull.InGreenStateMilliseconds,

vStateHourlyFull.InWhiteStateMilliseconds,

vStateHourlyFull.InDisabledStateMilliseconds,

vStateHourlyFull.IntervalDurationMilliseconds,

vStateHourlyFull.InPlannedMaintenanceMilliseconds,

vStateHourlyFull.InUnplannedMaintenanceMilliseconds,

vStateHourlyFull.HealthServiceUnavailableMilliseconds,

vManagedEntity.ManagedEntityGuid,

vStateHourlyFull.ManagedEntityMonitorRowId,

vManagedEntityTypeImage.Image,

vManagedEntity.ManagedEntityDefaultName,

vManagedEntityType.ManagedEntityTypeGuid

FROM

vStateHourlyFull INNER JOIN

vManagedEntity ON vStateHourlyFull.ManagedEntityRowId =

vManagedEntity.ManagedEntityRowId INNER JOIN

vManagedEntityType ON vManagedEntity.ManagedEntityTypeRowId =

vManagedEntityType.ManagedEntityTypeRowId INNER JOIN

vMonitor ON vStateHourlyFull.MonitorRowId =

vMonitor.MonitorRowId LEFT OUTER JOIN

vManagedEntityTypeImage ON vManagedEntity.ManagedEntityTypeRowId

= vManagedEntityTypeImage.ManagedEntityTypeRowId AND

vManagedEntityTypeImage.ImageCategory = N'u16x16Icon'

WHERE

(vManagedEntity.ManagedEntityRowId in (/\*List of Managed Entities\*/)

AND (vMonitor.MonitorSystemName = @MonitorName)

AND (vStateHourlyFull.DateTime &gt;= @StartDate)

AND (vStateHourlyFull.DateTime &lt; @EndDate)

AND (vStateHourlyFull.Date

BETWEEN DATEADD(day, -1, @StartDate) AND DATEADD(day, 1, @EndDate))

Performance

Performance data is stored directly in the OperationsManagerDW. Performance data is available in daily and hourly aggregates, and with calculated minimum, maximum, and average values, and in the standard deviation.

Perf.vPerfDaily

|  |  |
| --- | --- |
| Column | Datatype |
| Datetime | datetime |
| PerformanceRuleInstanceRowID | int |
| ManagedEntityRowID | int |
| SampleCount | int |
| AverageValue | float |
| MinValue | float |
| MaxValue | float |
| StandardDeviation | float |

Perf.vPerfHourly

|  |  |
| --- | --- |
| Column | Datatype |
| Datetime | datetime |
| PerformanceRuleInstanceRowID | int |
| ManagedEntityRowID | int |
| SampleCount | int |
| AverageValue | float |
| MinValue | float |
| MaxValue | float |
| StandardDeviation | float |

Perf.vPerfRaw

|  |  |
| --- | --- |
| Column | Datatype |
| Datetime | datetime |
| PerformanceRuleInstanceRowID | int |
| ManagedEntityRowID | int |
| SampleCount | int |

Query:

SELECT

vManagedEntityTypeImage.Image,

vPerfHourly.DateTime,

vPerfHourly.SampleCount,

vPerfHourly.AverageValue,

vPerfHourly.StandardDeviation,

vPerfHourly.MaxValue,

vManagedEntity.FullName,

vManagedEntity.Path,

vManagedEntity.Name,

vManagedEntity.DisplayName,

vManagedEntity.ManagedEntityDefaultName,

vPerformanceRuleInstance.InstanceName,

vPerformanceRule.ObjectName,

vPerformanceRule.CounterName

FROM

Perf.vPerfHourly INNER JOIN

vManagedEntity ON Perf.vPerfHourly.ManagedEntityRowId =

vManagedEntity.ManagedEntityRowId INNER JOIN

vManagedEntityType ON vManagedEntity.ManagedEntityTypeRowId =

vManagedEntityType.ManagedEntityTypeRowId LEFT OUTER JOIN

vManagedEntityTypeImage ON vManagedEntityType.ManagedEntityTypeRowId =

vManagedEntityTypeImage.ManagedEntityTypeRowId INNER JOIN

vPerformanceRuleInstance ON

vPerformanceRuleInstance.PerformanceRuleInstanceRowId =

Perf.vPerfHourly.PerformanceRuleInstanceRowId INNER JOIN

vPerformanceRule ON vPerformanceRuleInstance.RuleRowId =

vPerformanceRule.RuleRowId

WHERE

(vManagedEntity.ManagedEntityRowId in (/\*List of Managed Entities\*/))

AND(vPerformanceRule.ObjectName IN ('MEMORY'))

AND (vPerformanceRule.CounterName IN ('Available MBytes'))

ORDER BY vPerfHourly.DateTime

Discovery

There are two types of data that are discovered: objects and views.

Objects

Operations Manager discovers the objects that have discovery rules defined for them. Discovery data is then stored in the OperationsManager and in the OperationsManagerDW databases. Besides up-to-date discovery data, the OperationsManagerDW database also contains historical discovery data even after the respective objects are no longer discovered. Using that data, you can then search the OperationsManagerDW database for objects for which the OperationsManager database no longer has active monitoring data.

Views

This view describes which management group the object belongs to. This view allows you to specify a range in which to search for an object. A single object can have multiple rows in vManagedEntity, as it can be discovered and removed multiple times in a distinct management group.

|  |  |
| --- | --- |
| Column | Datatype |
| ManagedEntityManagementGroupRowId | int identity |
| ManagedEntityRowId | int |
| FromDateTime | datetime |
| ToDateTime | datetime |
| DWCreatedDateTime | datetime |
| DWLastModifiedTime | datetime |

This view displays objects that were discovered by Operations Manager. The OperationsManagerDW contains data about objects that are no longer being actively managed by a management group. There can be multiple rows per managed entity because an entity can be discovered and removed within a management group multiple times.

|  |  |
| --- | --- |
| Column | Datatype |
| ManagedEntityRowId | int identity |
| ManagementGroupRowId | int |
| ManagedEntityGuid | uniqueidentifier |
| ManagedEntityTypeRowId | int |
| FullName | ntext |
| Path | ntext |
| Name | ntext |
| DisplayName | ntext |
| ManagedEntityDefaultName | ntext |
| DWCreatedDateTime | datetime |
| TopLevelHostManagedEntityRowId | int |

Object Properties

Object properties are discovered by discovery rules from different management packs. The OperationsManagerDW stores the discovery data and tracks when the discovery finds changes and what old and new values are.

Events

Events are stored directly in the OperationsManagerDW database with their respective details and parameters. No aggregation or summarization is done.

Event.vEvent

|  |  |
| --- | --- |
| Column | Datatype |
| EventOriginId | uniqueidentifier |
| DateTime | datetime |
| EventPublisherRowId | int |
| EventChannelRowId | smallint |
| EventCategoryRowId | smallint |
| EventLevelId | tinyint |
| LoggingComputerRowId | int |
| EventNumber | bigint |
| EventDisplayNumber | int |
| UserNameRowId | int |
| RawDescriptionHash | uniqueidentifier |
| ParameterHash | uniqueidentifier |
| EventDataHash | uniqueidentifier |

vEventCategory

|  |  |
| --- | --- |
| Column | Datatype |
| EventCategoryRowId | smallint identity |
| EventPublisherRowId | int |
| EventCategoryId | smallint |
| EventCategoryTitle | nvarchar |
| LastReceivedDateTime | smalldatetime |

vEventChannel

|  |  |
| --- | --- |
| Column | Datatype |
| EventChannelRowId | smallint identity |
| EventChannelTitle | nvarchar |
| LastReceivedDateTime | smalldatetime |

vEventDetail

|  |  |
| --- | --- |
| Column | Datatype |
| EventOriginId | uniqueidentifier |
| RawDescription | ntext |
| RenderedDescription | ntext |
| EventData | xml |

vEventParameter

|  |  |
| --- | --- |
| Column | Datatype |
| EventOriginId | uniqueidentifier |
| ParameterIndex | tinyint |
| ParameterValue | nvarchar |

vEventUserName

|  |  |
| --- | --- |
| Column | Datatype |
| EventUserNameRowId | int identity |
| UserName | nvarchar |
| LastReceivedDateTime | smalldatetime |

vEventLoggingComputer

|  |  |
| --- | --- |
| Column | Datatype |
| EventLoggingComputerRowId | smallint identity |
| ComputerName | nvarchar |
| LastReceivedDateTime | smalldatetime |

Query:

SELECT

      vEvent.DateTime,

      vEventPublisher.EventPublisherName as 'EventSource',

      vEventLoggingComputer.ComputerName as 'Computer',

      vEventLevel.EventLevelTitle as 'Type',

      vEvent.EventDisplayNumber as 'EventID',

      vEventChannel.EventChannelTitle,

      vEventUserName.UserName,

    vEventDetail.RenderedDescription as 'EventDescription'

FROM

      Event.vEvent LEFT OUTER JOIN

      vEventUserName ON vEvent.UserNameRowId =

vEventUserName.EventUserNameRowId LEFT OUTER JOIN

      vEventCategory ON vEvent.EventCategoryRowId =

vEventCategory.EventCategoryRowId LEFT OUTER JOIN

      vEventPublisher ON vEvent.EventPublisherRowId =

vEventPublisher.EventPublisherRowId LEFT OUTER JOIN

      vEventLoggingComputer ON vEvent.LoggingComputerRowId =

vEventLoggingComputer.EventLoggingComputerRowId LEFT OUTER JOIN

      vEventLevel ON vEvent.EventLevelId = vEventLevel.EventLevelId LEFT OUTER JOIN

      vEventChannel ON vEvent.EventChannelRowId =

vEventChannel.EventChannelRowId LEFT OUTER JOIN

      Event.vEventDetail ON vEvent.EventOriginId = vEventDetail.EventOriginId

WHERE vEventLevel.EventLevelTitle = 'Error'

ORDER BY vEvent.DateTime, vEventLoggingComputer.ComputerName

Alerts

Alerts are stored directly in the OperationsManagerDW database with their respective details and parameters. No aggregation or summarization is done.

Alert.vAlert

|  |  |
| --- | --- |
| Column | Datatype |
| AlertGuid | uniqueidentifier |
| AlertProblemGuid | uniqueidentifier |
| AlertName | nvarchar(256) |
| AlertDescription | nvarchar(256) |
| Severity | tinyint |
| Priority | tinyint |
| Category | nvarchar(256) |
| ManagedEntityRowID | int |
| WorkflowRowId | int |
| MonitorAlertInd | bit |
| RaisedDateTime | datetime |
| SiteName | nvarchar(256) |
| RepeatCount | int |
| AlertStringGuid | uniqueidentifier |
| ParameterHash | uniqueidentifier |
| DBCreatedDateTime | uniqueidentifier |
| DWCreatedDateTime | uniqueidentifier |
| DWLastModifiedTime | uniqueidentifier |

Alert.vAlertDetail

|  |  |
| --- | --- |
| Column | Datatype |
| AlertGuid | uniqueidentifier |
| Owner | nvarchar(256) |
| TicketID | nvarchar(256) |
| CustomField1 | nvarchar(256) |
| CustomField2 | nvarchar(256) |
| CustomField3 | nvarchar(256) |
| CustomField4 | nvarchar(256) |
| CustomField5 | nvarchar(256) |
| CustomField6 | nvarchar(256) |
| CustomField7 | nvarchar(256) |
| CustomField8 | nvarchar(256) |
| CustomField9 | nvarchar(256) |
| CustomField10 | nvarchar(256) |
| DBLastModifiedDateTime | datetime |
| DBLastModifiedByUserId | nvarchar(256) |
| DWCreatedDateTime | datetime |

Alert.vAlertParameter

|  |  |
| --- | --- |
| Column | Datatype |
| AlertGuid | uniqueidentifier |
| ParameterIndex | tinyint |
| ParameterValue | nvarchar(255) |

Query:

SELECT

Alert.vAlert.AlertName,

Alert.vAlert.AlertDescription,

Alert.vAlert.Severity,

Alert.vAlert.Priority,

Alert.vAlert.Category,

Alert.vAlert.RaisedDateTime,

Alert.vAlert.RepeatCount,

vManagedEntity.FullName,

vManagedEntity.Path,

vManagedEntity.Name,

vManagedEntity.DisplayName,

vManagedEntity.ManagedEntityDefaultName

FROM Alert.vAlertDetail INNER JOIN

Alert.vAlert ON Alert.vAlertDetail.AlertGuid = Alert.vAlert.AlertGuid INNER JOIN

vManagedEntity ON Alert.vAlert.ManagedEntityRowId =

vManagedEntity.ManagedEntityRowId

WHERE vManagedEntity.ManagedEntityRowId in (/\*List of Managed Entities\*/)

AND Alert.vAlert.Severity = 1

Note: The queries above will return values for images which will allow you to visualize an object in a report.

Visualizing the objects shown using an image

Where is it used?

When an object is used in a report the image is queried from the vImage view.

How is it defined in the report?

<Image Name="ObjectImage">

<Sizing>Clip</Sizing>

<MIMEType>image/png</MIMEType>

<ZIndex>7</ZIndex>

<Source>Database</Source>

<Style>

<BorderStyle>

<Default>Solid</Default>

<Right>None</Right>

</BorderStyle>

<PaddingLeft>4pt</PaddingLeft>

<BorderColor>

<Default>DarkGray</Default>

</BorderColor>

<PaddingTop>2pt</PaddingTop>

</Style>

<Value>=Fields!Image.Value</Value>

</Image>

Where does the image come from?

The image is taken from a management pack