THE POWER TO KNOW.

# SAS® Real-Time Decision Manager 5.4

## Administrator's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS® Real-Time Decision Manager 5.4: Administrator's Guide*. Cary, NC: SAS Institute Inc.

**SAS® Real-Time Decision Manager 5.4: Administrator's Guide**

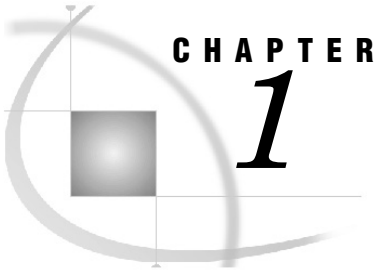


SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, February 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

# Contents



CHAPTER 1

# Introduction to SAS Real-Time Decision Manager



## Overview

SAS Real-Time Decision Manager software combines SAS analytics with business logic to deliver real-time decisions and recommendations to interactive customer channels. These channels include the Web, call center, point of sale (POS), ATM, and more. The product provides an extensible and service-oriented architecture. This scalable and fault-tolerant architecture makes it ideal for continuous operation in environments with high-transaction volumes.

User-defined processes that contain business rules, SAS analytics, and actions are called decision flows. Decision flows are surfaced as Web services that field requests from customer-facing channels in real-time. An administrator performs the following tasks:

- Controls which decision flows are operational at any given time
- Promotes decision flows from development to test to production environments
- Configures and maintain the SAS Real-Time Decision Manager operational environment, ensuring that appropriate resources are available within each environment
- Monitors the distributed, fault-tolerant hardware environments in which SAS Real-Time Decision Manager operates

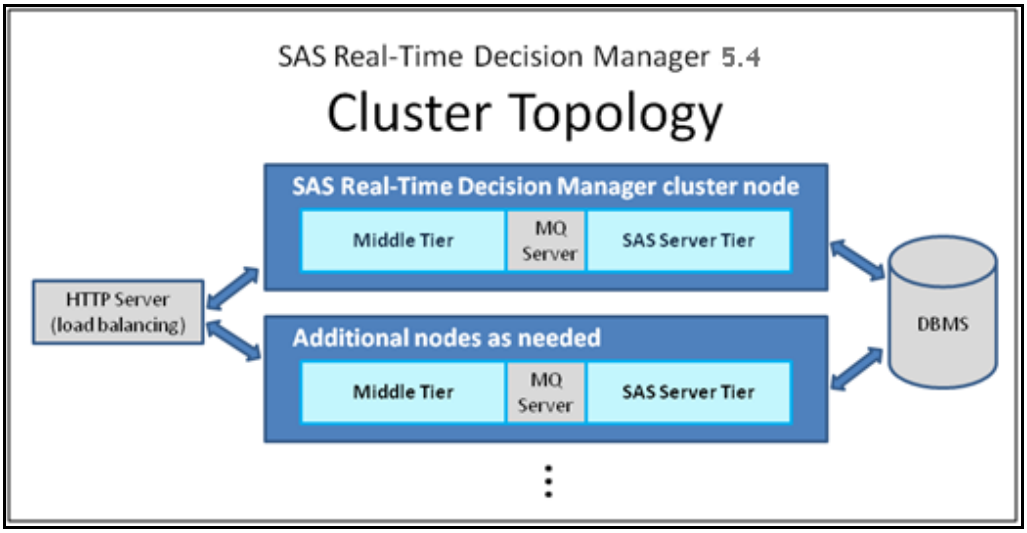
# Concepts

## Scalability and Failover

SAS Real-Time Decision Manager is designed as an enterprise-class distributed system to achieve high availability with no single point of failure. At the same time, the system is centrally configured using SAS Management Console and SAS Metadata Repository.

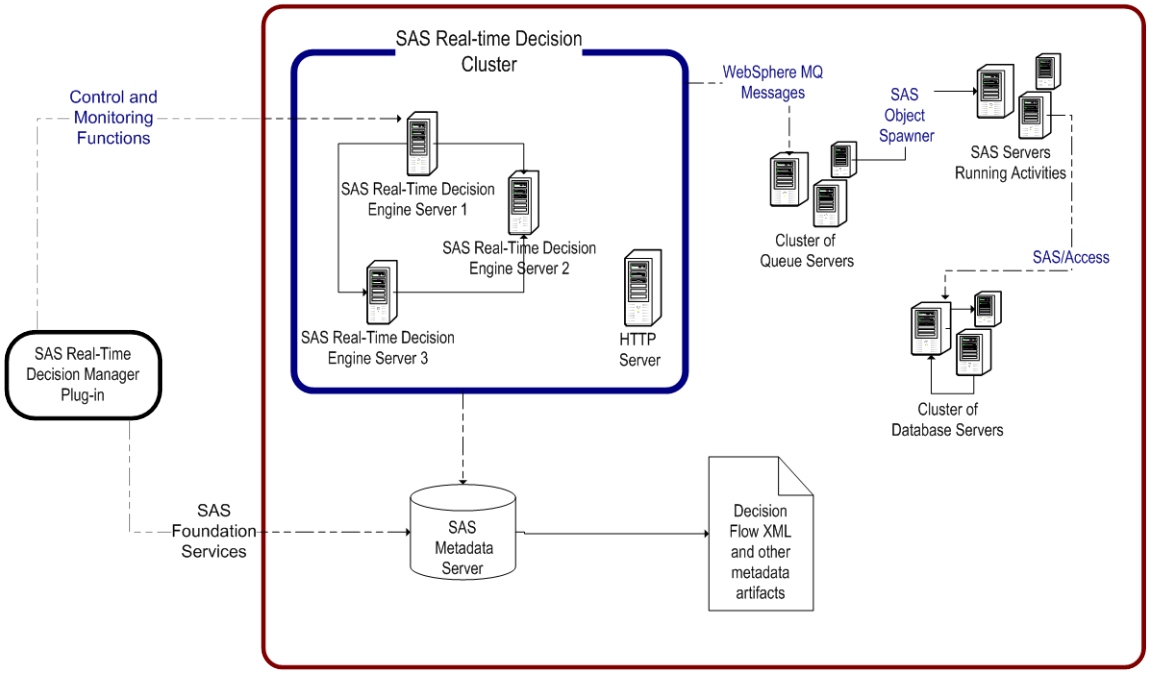
The SAS Real-Time Decision Manager engine is deployed within a J2EE application server container. The clustering and load balancing capabilities of the application server combine with the SAS Real-Time Decision Manager threaded architecture to enable parallel execution. At any point in time, servers can be removed from or added to the cluster without stopping the application (for example, if a server fails and re-starts). To support this operation without human intervention, all configuration information that is required to initialize and operate the system is made available in a fail-safe manner within a cluster-wide lateral cache.

In addition to the middle tier, SAS Real-Time Decision Manager includes a configurable cluster of SAS servers that bring advanced analytics to the process of making business decisions in a failsafe manner.




## Deployment Topology

The illustration below shows the various logical components of SAS Real-Time Decision Manager when deployed in a production environment. The clustering capabilities of this enterprise application provide a highly scalable environment designed to deliver timely real-time analytical decisions.




The production environment consists of either a single instance or multiple instances of the following servers, depending on performance and availability requirements.

- **SAS Metadata Servers** contain artifacts such as global variables, SAS activities, events, and lightweight metadata objects that are pointers to decision flows in the content repository.
- **SAS Real-Time Decision Engine Servers** are configured in an application server cluster. These servers execute the decision flows that provide the real-time analytical decisions and actions.
- **SAS Servers** primarily run the SAS activities using SAS Connection Servers. A SAS Connection Server consists of a SAS Object Spawner configured as an MQ polling server, a special SAS activity framework program, and a supporting macro library. A SAS activity can be configured to run any SAS program, although only high-performance programs are recommended. Appropriate SAS licenses must be obtained for the products used.
- **HTTP Servers** are one example of the many available load balancing solutions for the real-time decision cluster enterprise. Using Service-Oriented Architecture (SOA) integration through Web services, HTTP servers can be used as integration points between external applications and a SAS Real-Time Decision Manager cluster.
- **Message Queue Polling Servers** manage messages that pass between the SAS Real-Time Decision Manager Engine cluster and the SAS servers, where advanced analytics and user-written procedures execute.
- **Application Servers** may be configured as a cluster and used for deployment of the SAS Real-Time Decision Manager engine server.
- **Database Servers** store data.

The SAS real-time decision cluster enterprise makes extensive use of open standards to simplify integration and maximize interoperability.

## A Typical Configuration

A typical installation consists of development, test, and production environments, although the number of environments is configurable to accommodate internal approval process standards. Decision flows are created and functionally tested in the development environment by business users. When a business user is satisfied that a decision flow is ready for deployment, an administrator promotes the flow to either a test or production environment. A test environment is optional and can be used to conduct performance testing on decision flows in an environment similar to the production environment. The production environment serves "live" channels or customer-facing systems. Each environment includes a repository of decision flows, their building blocks, and other resources.

The SAS Management Console import/export functionality is used to promote artifacts from one repository to another repository. In this case, decision flows and other artifacts are promoted between development, test, and production environments.

The SAS Real-Time Decision Manager plug-in also operates on these repositories and is used to monitor and control SAS Real-Time Decision Manager runtime systems from a central location.

After a flow is promoted, the SAS Real-Time Decision Manager plug-in can be used to activate the flow, putting it into production.

## Development Environment

The development environment enables business users to create, test, edit, and delete decision flows. The SAS Real-Time Decision Manager Design Server provides this functionality through a Web service API. SAS Customer Intelligence Studio provides a user-friendly drag-and-drop interface, and uses the SAS Real-Time Decision Manager design server to execute the above functionality on the users' behalf.

Decision flows and their building blocks (events, activities, global variables, and system resources) are stored in a repository. Each repository resides in SAS Metadata Server. Repositories are managed by the SAS Real-Time Decision Manager plug-in.

A development environment contains:

- SAS Customer Intelligence GUI and common components
- SAS Real-Time Decision Manager Design Server
- IBM WebSphere Application Server
- IBM WebSphere MQ
- SAS Metadata Repository
- SAS Management Console

- SAS Real-Time Decision Manager Engine Server (for decision flow testing)
- SAS Server (for decision flow testing)

### Test and Production Environments

From a software topology perspective, the test and production environments are identical. The production environment provides the capabilities and performance required for 24/7/365 operation.

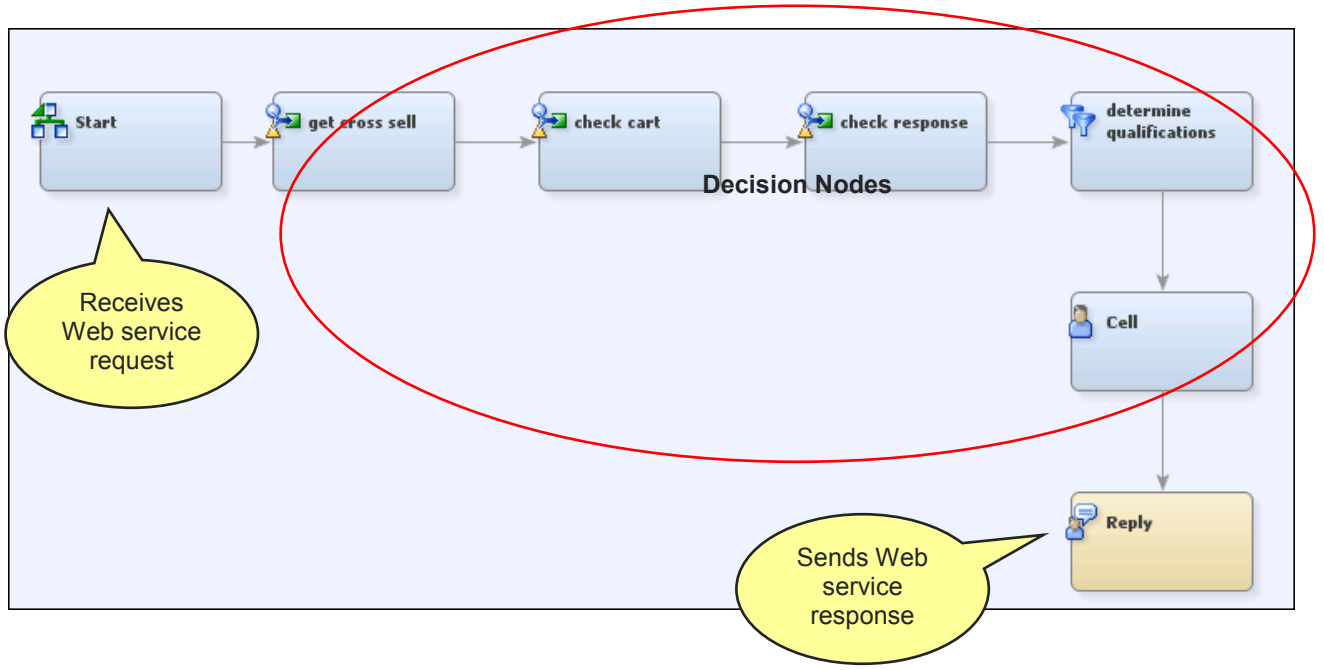
As with the development environment, decision flows and their building blocks are stored in a repository. Repositories and their contents are managed by the SAS Real-Time Decision Manager plug-in and the SAS Customer Intelligence plug-ins. An important function of the SAS Real-Time Decision Manager plug-in (within the test and production environments) is to activate or deactivate decision flows. Activating or deactivating decision flows either connects or disconnects decision flows with the live channels.

A test or production environment contains:

- SAS Real-Time Decision Manager Engine Server cluster
- IBM WebSphere Application Server and Deployment Manager with HTTP Server (WebSphere Network Deployment Environment)
- IBM WebSphere MQ
- SAS Server cluster
- SAS Metadata Repository
- SAS Management Console
- A 3rd party database management system

## Example: The Decision Flow

Consider, for example, a retail business, where SAS Real-Time Decision Manager supports a Web site and an inbound call center. Many decision flows might be deployed to process the various requests that originate from those systems. The following scenario describes a simple example of a cross-sell offer.




When a customer calls the call center and purchases a product, the customer service representative (CSR) wants to make the best possible cross-sell offer. When the CSR enters the purchase information, the call center application sends a Web service request to SAS Real-Time Decision Manager, requesting the best cross-sell offer to present.

Each active decision flow handles one Web service request type. This guarantees that when a cross-sell Web service request is received, the appropriate decision flow processes it. Note that many copies of each decision flow can process multiple requests concurrently and are available to field a high volume of transactions.

In SAS Real-Time Decision Manager parlance, a Web service request is known as an event. Each decision flow begins with a Start activity. When the cross-sell event is received, the Start activity places the relevant request data into a block of in-memory variables known as process variables. In the example, the request data includes the customer's ID and shopping cart items.

The decision flow continues to execute, processing one activity after another, until a Reply activity is reached. The Reply activity sends the results of the decision flow back to the call center via the Web service reply message.

Each activity in a decision flow performs an action. An activity reads the data needed to perform its action from the process variables and it writes the results of that action back to the process variables. In this way, downstream activities can use the outputs of upstream activities as inputs. In the previous example, these are the actions that are performed:

1. Get the request data (Start activity).

2. Retrieve the best cross-sell offer based on the customers' primary purchase. This step could employ any number of SAS analytical techniques, such as scoring the customer with a propensity-to-buy predictive model.
3. Verify that the recommended cross-sell product isn't already in the customers' basket.
4. Check the response history to make sure that the customer has not previously received a cross-sell offer and rejected it.
5. Verify that the customers demographic information make her a good candidate for the offer.
6. Record the offer history for future real-time use or offline analysis.
7. Reply with the offer.

More complex decision flows may include branching rules, where the sequence of activity execution is controlled by a set of conditional expressions.

## Life Cycle of a Decision Flow

To deploy a decision flow into production, it must be developed, tested, promoted to a production system, and activated. Let's briefly examine each of these stages of the decision flow life cycle. Promotion and activation procedures are described in detail in the Common Operations section later in this guide.

### Development and Testing

End users develop decision flows using SAS Customer Intelligence Studio. This business-user friendly interface allows decision flows to be constructed by dragging and dropping activities from a palette. It also supports the development of decision flow tests. See *SAS Real-Time Decision Manager 5.4 User's Guide* for details.

A significant advantage of the activity model is that business users do not need to understand the complex algorithms used. Rather, business users need only to understand how each activity either selects or transforms the data. However, statisticians and other analysts have full access to the underlying algorithms and can change or replace them as needed.

#### *Promotion*

At a minimum, a SAS Real-Time Decision Manager deployment will include a development environment and a production environment. Optionally, one or more test environments may be included as well. In this context, a test environment is just like a production environment but is not connected to live channels. The kind of testing performed depends on company policy. Examples include performance testing and verifying flow results over a large set of sample inputs.

When a business user marks a decision flow for deployment, the flow is persisted in a SAS Real-Time Decision Manager repository. If a flow is marked for deployment more than once, then the new copy of the flow overwrites any previous copy. Marking a flow for deployment in SAS Customer Intelligence Studio is similar to issuing a `Save` command. Once the flow is persisted, the administrator takes control of the decision flow. The administrator works primarily within SAS Management Console.

Each environment (development, test, and production) has an associated repository. When an end user marks a flow for deployment, SAS Customer Intelligence Studio calls the SAS Real-Time Decision Manager design server that stores the flow in the development repository.

To promote a decision flow, the administrator exports the flow from the development repository and imports it into a test or production repository (see Promoting Decision Flows for details).

### *Activation*

Each decision flow in a test or production environment is either active or inactive. Inactive flows are not loaded by a SAS Real-Time Decision Manager engine server. To put a flow into production (or make it ready for testing in a test environment) the administrator must activate it. To remove a flow from production, the administrator deactivates it. See Activating Flows below for details.

## Artifacts

Decision flows, their building blocks, and associated SAS Real-Time Decision Manager *artifacts* are described here.

A set of activities and system resources are provided with the product and are typically configured by on-site SAS support personnel at the time your system is installed. On-site SAS support personnel will also work with your IT department to define the events that are appropriate to your processing needs. The SAS Real-Time Decision Manager plug-in for SAS Management Console provides advanced functions that support the creation, editing, and deletion of system resources (see Repository Management for details). Other types of artifacts are created or deleted using the SAS Real-Time Decision Manager Design Server API's. SAS Customer Intelligence Studio and SAS Customer Intelligence plug-in to SAS Management Console utilize SAS Real-Time Decision Manager Design Server API's for this purpose.

### Events

Each request for a decision is presented to the system as an event. These events and their associated decision flows are exposed to external clients as Web services. An event definition specifies a request message format and a reply message format. Events that are designed only to receive information may omit the reply message. An event makes up the contract between an external system and a decision flow, specifying the types of information contained within the request and reply. Typically, your IT department will integrate Web service requests into your systems, and on-site SAS support personnel will define the events that field those requests.

### Activities

An activity is a component of business work such as computing a credit score, or performing a market basket analysis. Activities are represented as the nodes of a decision flow diagram. Each activity contains a set of actions. For example, the general I/O activity contains the actions READ, and WRITE (which includes INSERT and UPDATE). Each action contains a set of inputs and outputs that are mapped to process variables. The activities provided with SAS Real-Time Decision Manager 5.4 contain a rich set of functionality appropriate for inbound marketing automation. The activities

within a flow can execute sequentially or concurrently as specified by the containing flow.

## Decision Flows

A decision flow (also called a flow) defines the set of decisions and actions to take when a third-party system such as a Web site or a call center sends a request to SAS Real-Time Decision Manager. A decision flow is comprised of activities and business logic that determines the order in which the activities are processed. Each individual type of request has one decision flow that is associated with it. Multiple copies of each decision flow can process multiple requests concurrently and are available to field a high volume of transactions. Decision flows are created by end-users in SAS Customer Intelligence Studio 5.4.

## Process Variables

Process variables are a set of in-memory typed variables that hold the results of activity actions during flow execution. Process variables enable downstream activities to use the results of upstream activities. For example, a Start activity might write the customer ID that is received from an inbound event to a process variable. Subsequently, a Score activity might be configured to run its Propensity action, which takes the customer ID process variable as input and writes a propensity-to-buy score to another process variable. Following this, the new value of the score might cause a decision activity to branch, and so on.

## System Resources

System resources are artifacts that provide activities with access to external resources within their environment, such as relational databases, SAS servers, or messaging middleware. For example, many activities rely on running a SAS program to produce results. Because flows execute in a Java 2 Enterprise Edition (J2EE) servlet container in the middle tier, these activities must communicate with SAS back-end servers. A SAS Connection is a type of system resource that provides the information that is needed to facilitate such communications.

The fact that activities *reference* system resource information (rather than *contain* system resource information) makes flows portable between systems. SAS Real-Time Decision Manager supports configurable development, test, and production environments. Typically, the set of back-end SAS servers that is used by development and production environments is different. System resources enable the correct set of servers to be used in each environment without modification to the decision flow.

## Global Variables

Global variables are used to tune the behavior of flows at execution time. For example, by modifying the value of a global variable that contains a customer risk threshold, the boundary between a medium-risk customer versus a high-risk customer can be adjusted at run time without changing any expressions or re-deploying the flow. See Managing Global Variables for details.

Unlike process variables, global variables are read-only with respect to flows and are cluster scoped rather than flow scoped. The value of a global variable affects the

behavior of every flow within a SAS Real-Time Decision Manager engine server cluster that references the global variable.

## Sub-flow

A sub-flow is a flow that is invoked by another flow. The purpose of sub-flows is to support recursive composition, whereby complex flows can be produced by combining simpler, easier-to-understand flows that perform a targeted set of tasks.

There are no distinctions between flows and sub-flows other than the fact that sub-flows are called by other flows. Sub-flows are event-driven like any other flows. To invoke a sub-flow, the user includes a sub-flow activity which enables the user to select the event that drives the desired sub-flow, and to map the event request and reply fields to process variables in the parent flow.

A sub-flow within a particular flow might execute sequentially or concurrently, depending upon how the parent flow is configured.

## ConcurrentWait Node

This node causes the main flow of execution to wait until ALL preceding concurrent nodes have finished execution. In case a concurrent node throws an exception, the following ConcurrentWait node captures it and throws it as a fault. The wait in a ConcurrentWait node is timed. If a concurrent node does not complete execution in the given time, the following ConcurrentWait node throws a timeout fault.

If there are no preceding concurrent nodes, then a ConcurrentWait node does not do anything. It is possible to have more than one ConcurrentWait node in a flow. Only those concurrent nodes that are not waited on by a preceding ConcurrentWait node are waited on by the later ConcurrentWait nodes.

## Fault Path

Because many operations that execute in process-based systems cannot be rolled back (such as sending a message to a third-party system), when an error occurs, such systems typically rely on compensation actions rather than on atomic transactions.

A fault path is a special branch of a flow that is taken when an error condition is detected. Examples of error conditions include receipt of bad data or lost contact with a database. Some users might specify that a reply be returned when any error occurs. Advanced users might create a fault path that performs compensating actions based on error type. A fault path can be associated with one, many, or all exception types. A fault path must end with a Reply activity and cannot be rejoined to the main flow.

## Concurrent Execution of Nodes

Activity nodes and sub-flow nodes can have an optional Boolean concurrent attribute that indicates whether they should be executed concurrently. If this attribute is true, then these nodes are scheduled for execution on a thread in parallel with the main thread of execution. If the attribute is false, then the nodes execute as in earlier versions of Real-Time Decision Manager. The order of execution of concurrent nodes is indeterminate.

There are three sub-tasks that take place in activity and sub-flow nodes:

1. Process variable values are copied to activity variables or event variables for activity and sub-flow nodes respectively.
2. The actual activity or event is executed.
3. Activity variable or event variable values are copied back to process variable values.

If the nodes are marked concurrent, then the task:

1. Takes place when the node executes.
2. Is scheduled for execution immediately afterwards.
3. Is executed in a ConcurrentWait node.

There are several implications of this:

1. If there is no ConcurrentWait node following a concurrent node, the output of the concurrent node is not captured as process variable values. Faults and timeouts are also ignored. However, the node will execute. This could be used for asynchronous execution.
2. The copying back of values to process variables takes place in the main execution thread. However, if the same process variables are referenced for output in other concurrent nodes, the last node is executed.
3. In case of an exception, a fault or timeout in any concurrent node preceding the ConcurrentWait node, no process variables are updated.

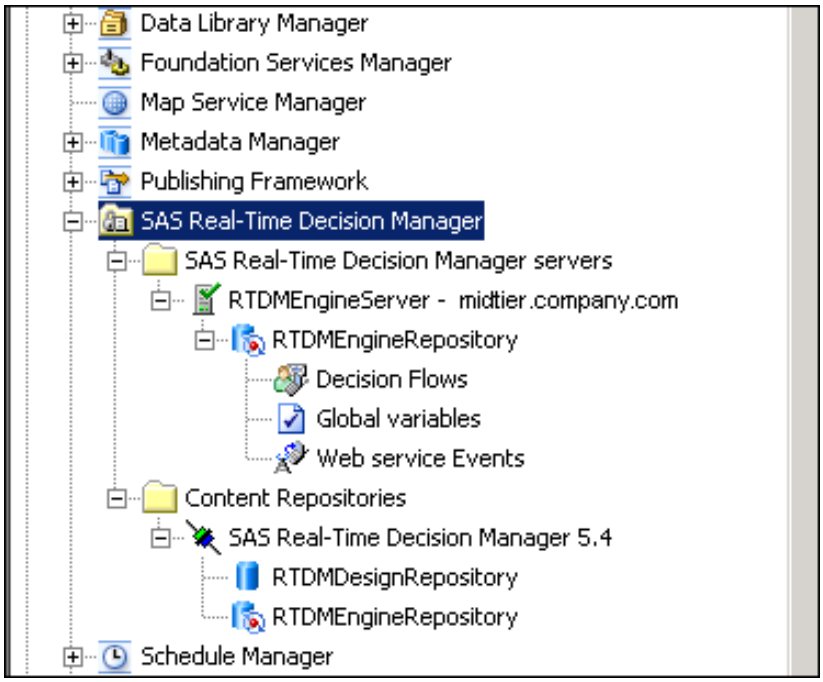
## SAS Real-Time Decision Manager Plug-in for SAS Management Console

Most administrative functions are carried out using the SAS Real-Time Decision Manager plug-in. This plug-in is specifically designed for users who need to update, administer, control, and monitor a test or production real-time decision cluster. The plug-in can be used from any client machine that runs SAS Management Console. Users of this plug-in are system administrators, system operators and performance analysts.

The plug-in is divided into two folders:

- The **SAS Real-Time Decision servers** folder provides control of the SAS Real-Time Decision Manager Engine server clusters, allowing an administrator to activate and deactivate decision flows and to change the values of global variables.

- The **Content Repositories** folder enables an administrator to manage SAS Real-Time Decision Manager repositories and their contents.




Icons are used to represent the status of the real-time decision cluster as well as the type of logical repository that is referenced by the icons. The meaning of each icon is as follows:


Indicates that the plug-in is connected to the SAS Real-Time Decision Manager cluster MBean Server.


Indicates that the plug-in cannot connect to the SAS Real-Time Decision Manager cluster MBean Server. This typically means that the cluster is not running.

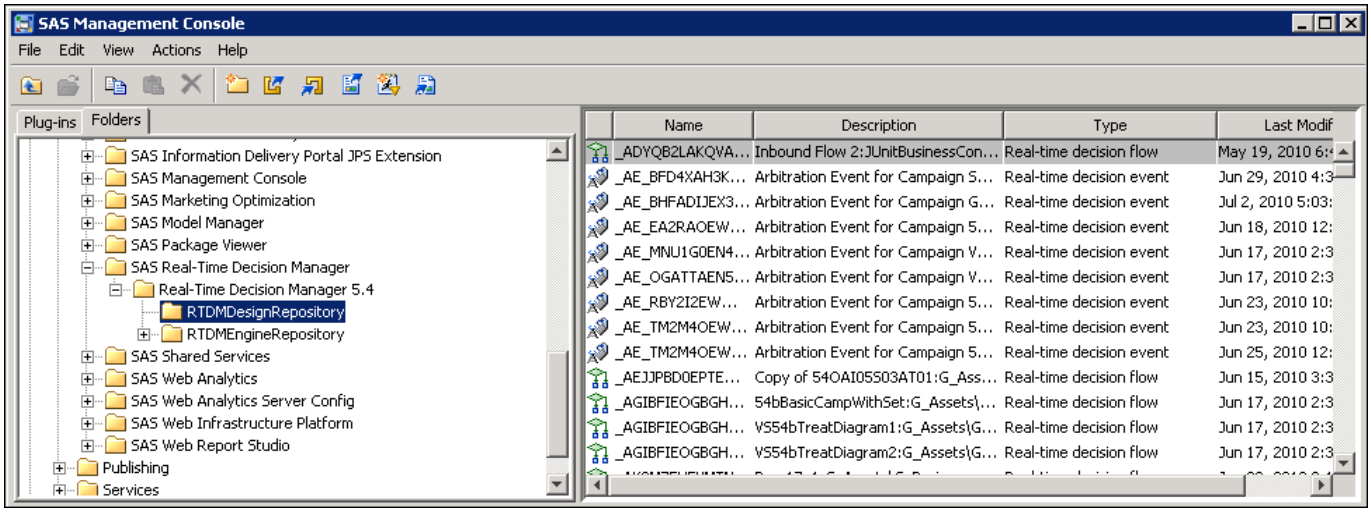

Indicates that the logical repository is a production repository.


Indicates that the logical repository is a test repository.

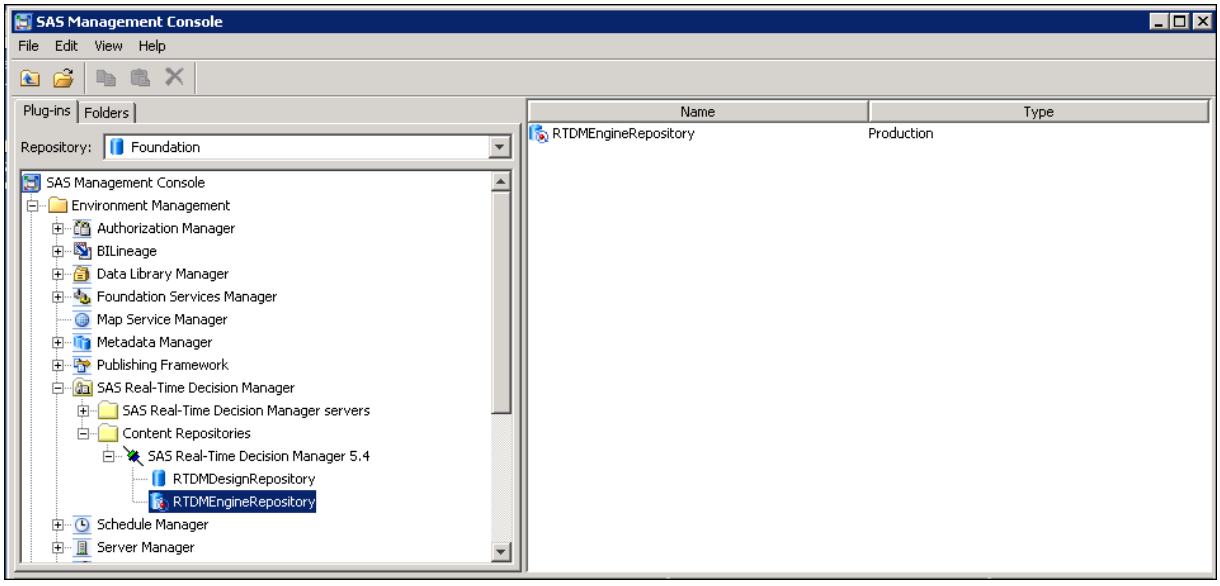

Indicates that the logical repository is a development repository.

## SAS Real-Time Decision Manager Repository

A SAS Real-Time Decision Manager repository can be viewed in SAS Management Console by using either the **Folders** view or SAS Real-Time Decision Manager plug-in. In the Folders view, each artifact has an associated name, description, type, and modification date.

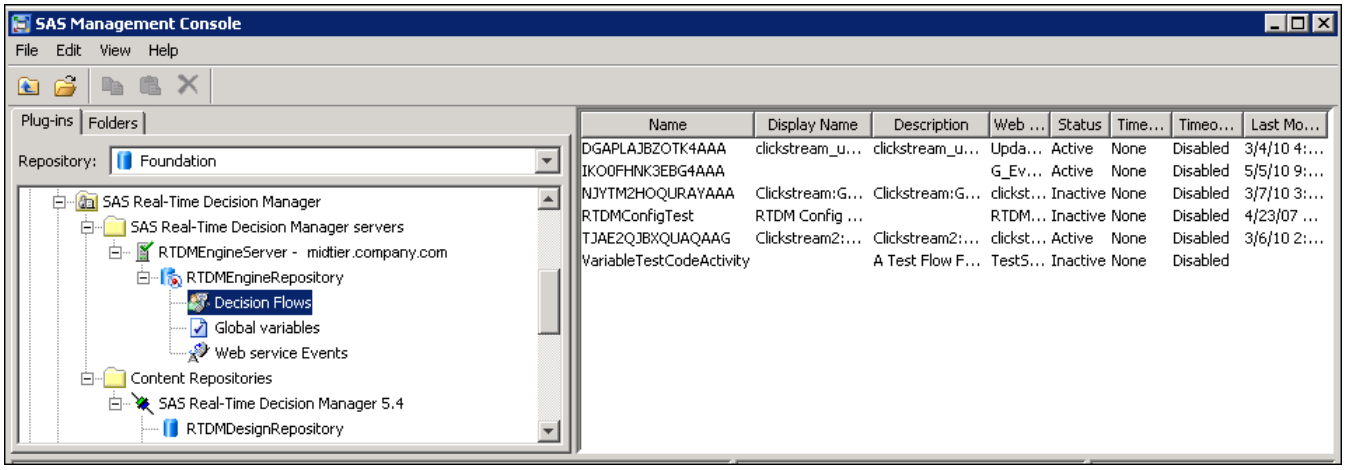



In the SAS Real-Time Decision Manager plug-in, the folder hierarchy is slightly different. It shows a context-sensitive view of the repository and provides product-specific functionality. The Folders tab displays a non-context-sensitive view that works with any product. Although rendered differently, both plug-ins display the same data.




When SAS Real-Time Decision Manager artifacts are promoted between development, test, and production environments, the files are copied from one repository to another.

The plug-in displays information about the artifact by reading and interpreting the product-specific metadata. In the screen shot below, when **Decision Flows** is selected, the flow name, display name, description, associated event, status (active or inactive), timeout value, timeout status (enabled or disabled), and the last modified date are displayed.



CHAPTER 2

# Common Operations

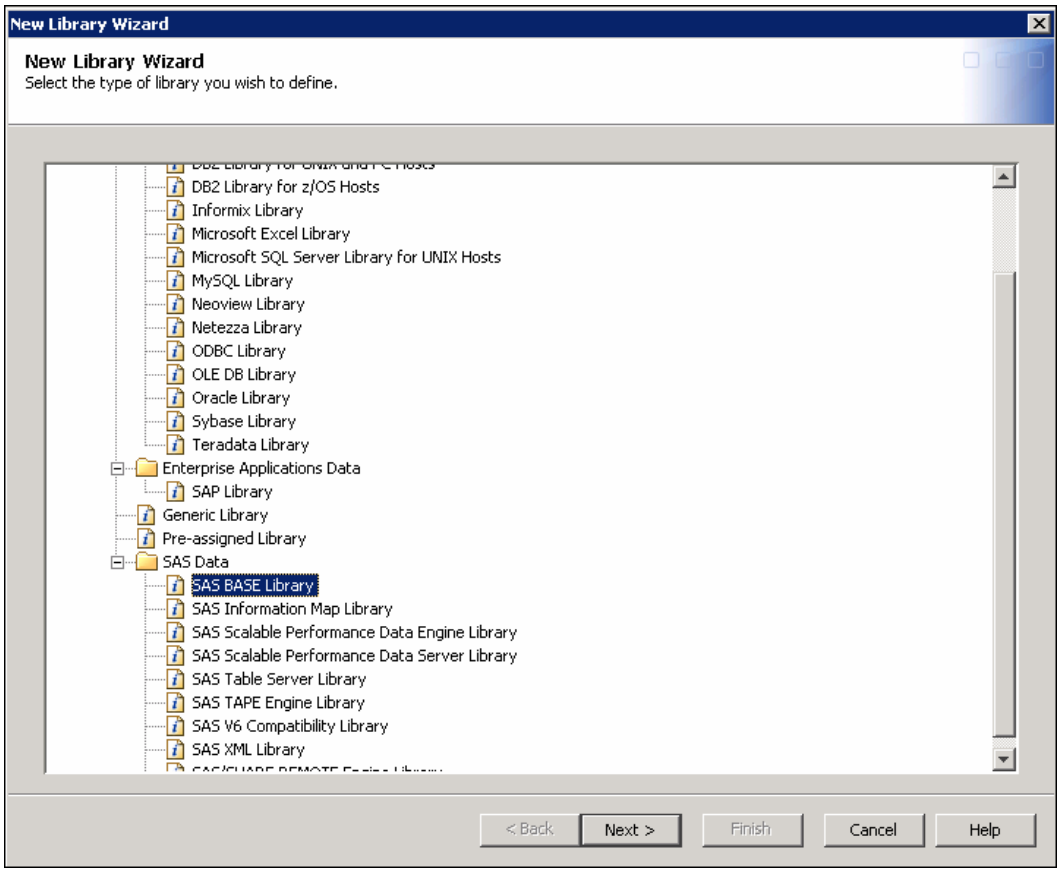


## Create the SAS Libraries for SAS Real-Time Decision Manager Servers

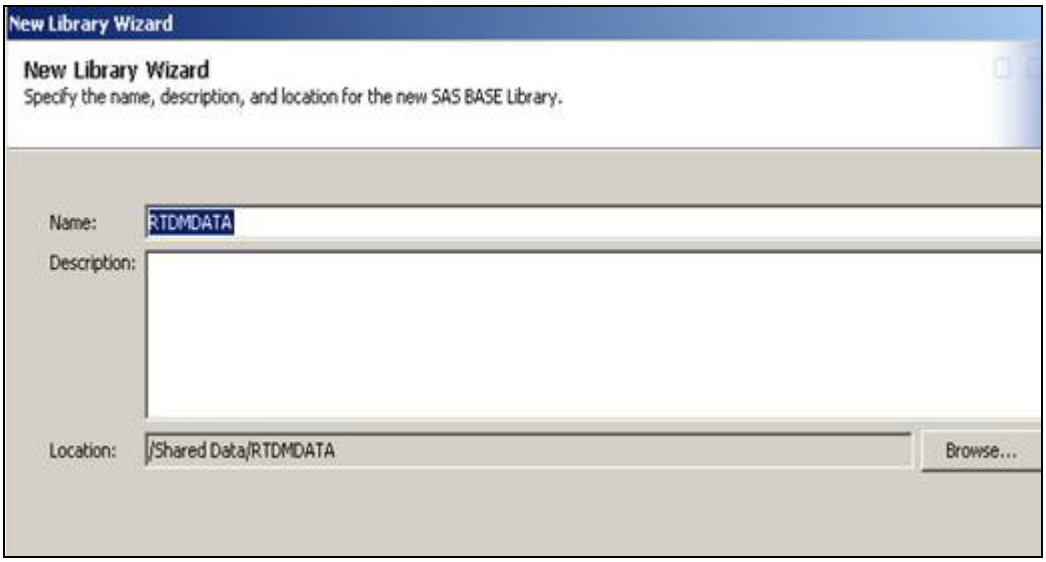
Create any SAS Real-Time Decision Manager server libraries (and other libraries) that will be used by your activities by following these steps:

1. In SAS Management Console, right-click the Libraries folder in the Data Library Manager plug-in and select **New Library**.


2. In the New Library Wizard, select **SAS BASE Library** and click **Next**.

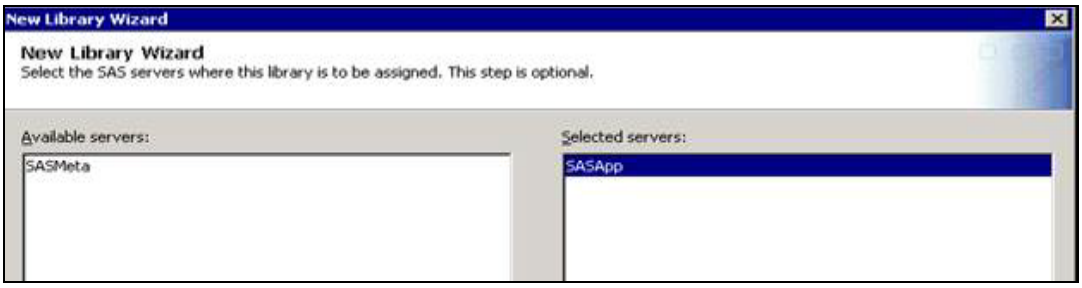



3. Type a library name such as RTDMDATA.

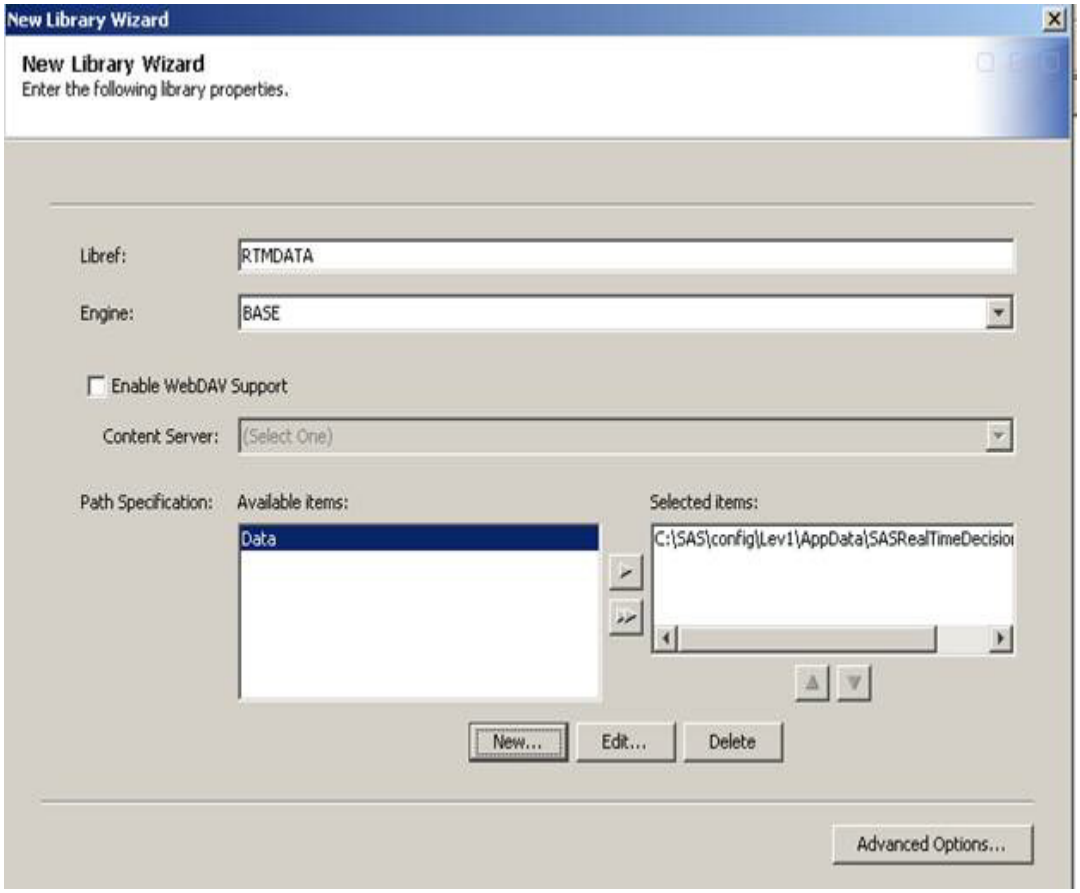



4. Select **Browse.**
5. Click the folder with the asterisks at the right top corner of the **Select a Location** dialog box.

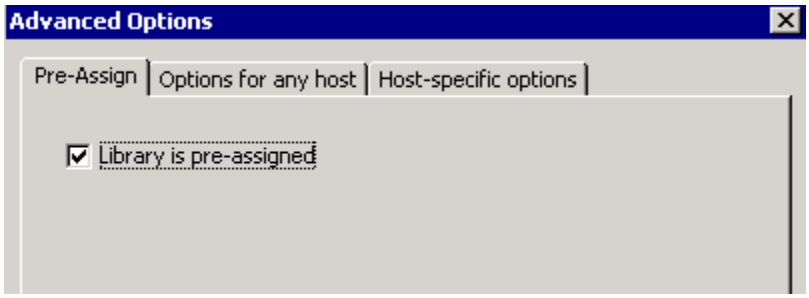
6. Enter a new folder called RTDMDATA, highlight the folder and select **OK**. Click **Next.**
7. Select the available server called **SASApp**. Click **Next**.




8. Enter the name of the libref as RTDMDATA and select **New** under **Path Specification**.
9. Select the **Browse** button. Logon as `sasdemo` or as any user ID that is defined in the metadata and that has a logon batch assigned on the machine where you are logging into.
   **Note:** Sometimes user ID `sastrust` works as well.
10. In the **Browse** dialog box, navigate to the directory where your SAS Tables are stored and click **OK**. The new path is displayed in the **Selected items** field.




11. Select Advanced Options.




12. Check the check box **Library is pre-assigned**, and click **OK**.
13. Click **Next** and **Finish**.

14. Right-click the created library and select **Properties**.

15. On the **Authorization** tab, make sure that you grant the correct permissions (change from *Deny*).

16. To create the CONNLIB library, repeat the previous steps.

# Register Source Tables with the New Library

Take the following steps to grant permissions to other users to use the data in the Share Data/RTDMDATA folder:

1. On the Folders tab, expand the Shared Data folder. Right click the RTDMDATA folder and select Properties.

2. On the **Authorization** tab, grant SASUSERS Read, Write Metadata, and Write permissions. Select **OK**.

   **Note:** When you select WriteMetadata permissions, you should also automatically receive WriteMemberMetadata permissions. This additional permission is acceptable and required. For testing purpose, you can give public users full access.

3. Using the SAS Management Console Data Library Manager, import the table definitions from the new library that you created.
4. On the Plug-ins tab, right-click the new library, and select **Register Tables.**
5. You do not need to make any changes on the first page. Select **Next.**
6. From the list of tables, select the tables that you want to register.
7. Select **Next** and **Finish**.

# Verify that the SASApp Server is Assigned to the Object Spawner

An object spawner should have been created when you installed SAS 9.2. Make sure that the SASApp server was assigned to the object spawner.

1. Right-click Object Spawner – Hostname and select its properties.
2. In the properties dialog box, select the Servers tab. Verify that the SASApp workspace server is listed under Selected Servers

# Promoting Decision Flows

You typically promote a flow from a *development* environment to a *test* environment, or from a *test* environment to a *production* environment (see Life Cycle of a Decision Flow above for more information). However, flows and other artifacts can be promoted from any SAS Real-Time Decision Manager repository to any other SAS Real-Time Decision Manager repository. If you are unfamiliar with repositories, see SAS Real-Time Decision Manager Repository above.

## Promotion Rules

Note: It is highly recommended that you promote only Flows and Variables under normal circumstances.

**As a general rule, resources should not be promoted.** System resources define the way SAS Real-Time Decision Manager interacts with external systems. Since those systems and interactions are different in a production environment than in a development environment, promoting a resource can have undesirable consequences.

**Do not promote an active artifact.** An active artifact contains the list of decision flows that are active in an environment. That list should never be copied from one environment to another. Because the plug-in the **Folder** view of SAS Management Console performs the promotion and is application-independent, it does not prevent an active artifact from being copied. Therefore, this restriction must be enforced as a best practice.

**Do not overwrite an active flow.** If you overwrite an active flow, then the engine will not be notified that the flow changed in the repository. Instead, deactivate the flow in the target system, promote it, and activate it. These steps will cause the engine to load the updated flow. Note that the active/inactive state of a flow is not carried along with the flow when the flow is promoted.

SAS Real-Time Decision Manager ships with a rich set of activities. If your organization develops a new activity that extends SAS Real-Time Decision Manager functionality, that activity may be promoted. Any system resources that are referenced by the new activity should be created in the target repository before flows that use the activity are activated. Activity promotion, if it occurs at all, should be a rare occurrence.

If you define a new event (and create a corresponding Web service request that calls SAS Real-Time Decision Manager), then as long as no event with the same name already exists in the target repository, it is safe to promote that event. If you overwrite an existing event, then any active flows or sub-flows that use the event might fail. To update an existing event, make sure that all flows using the original version of the event are deactivated first.

## Promotion Procedure

Promotion is accomplished in SAS Management Console by using the import/export functions from **Folder** view. Promotion consists of exporting artifacts from one repository and importing them into another repository.

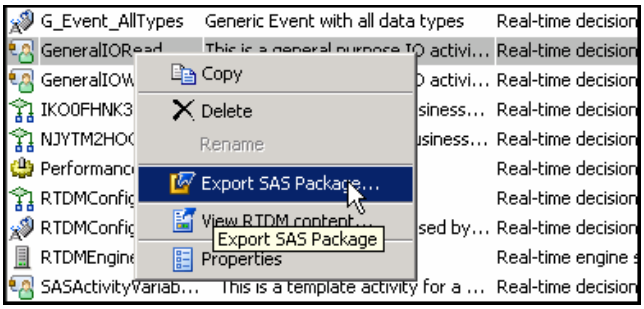
The artifact types that you can export are activity, flow, variable, event, and resource.

**CAUTION:** The **Folder** view in SAS Management Console does not restrict where artifacts can be imported. To avoid unpredictable results, follow these rules:

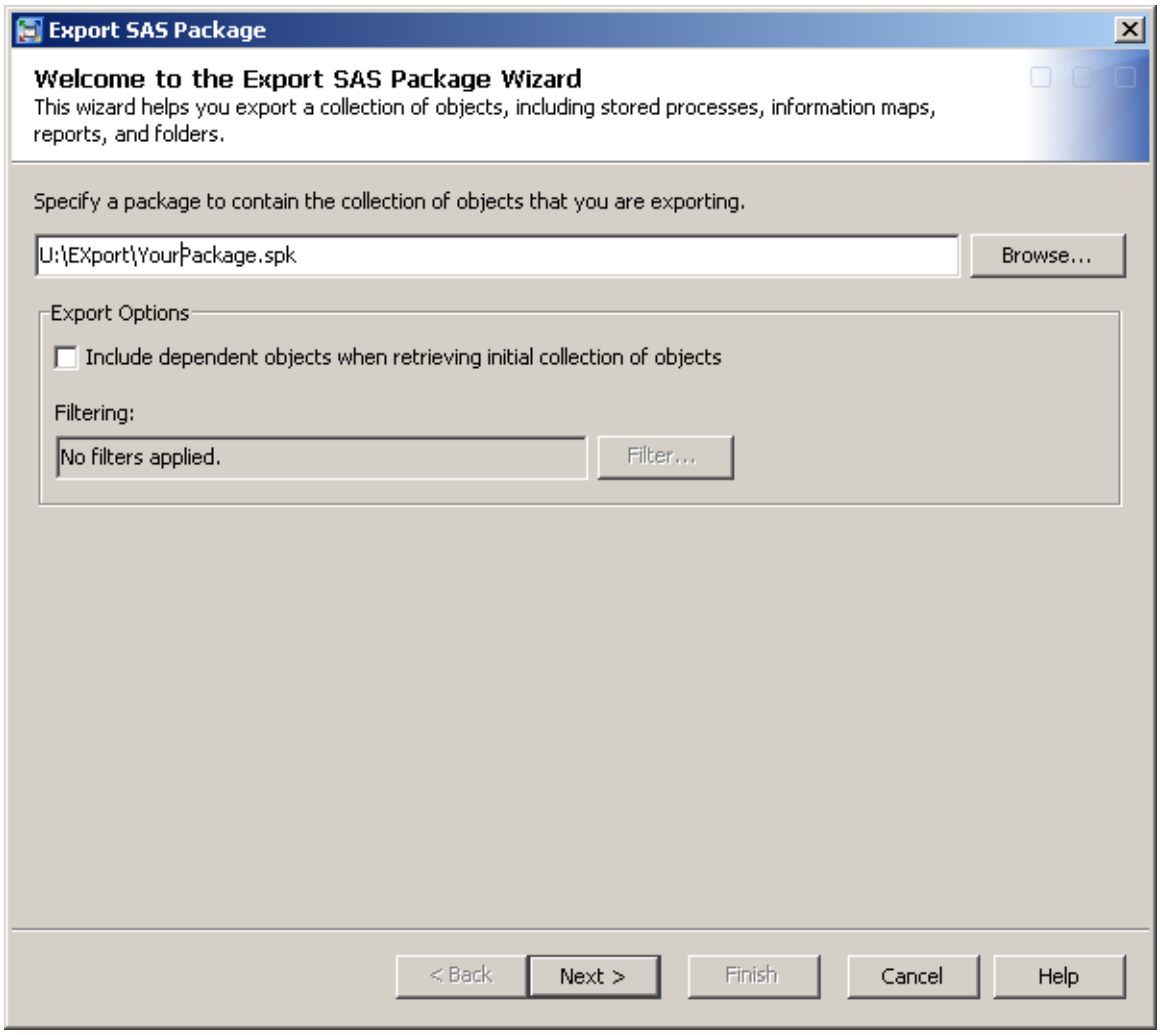
- Always export from an individual artifact.
- Always import to a repository folder.

The example that follows illustrates the promotion of a flow from the development repository “RTDMEngineRepository” to the production repository “YourProductionRepository.” Although both repositories are contained by the same content mapped folder and application context in the example, this is not required.

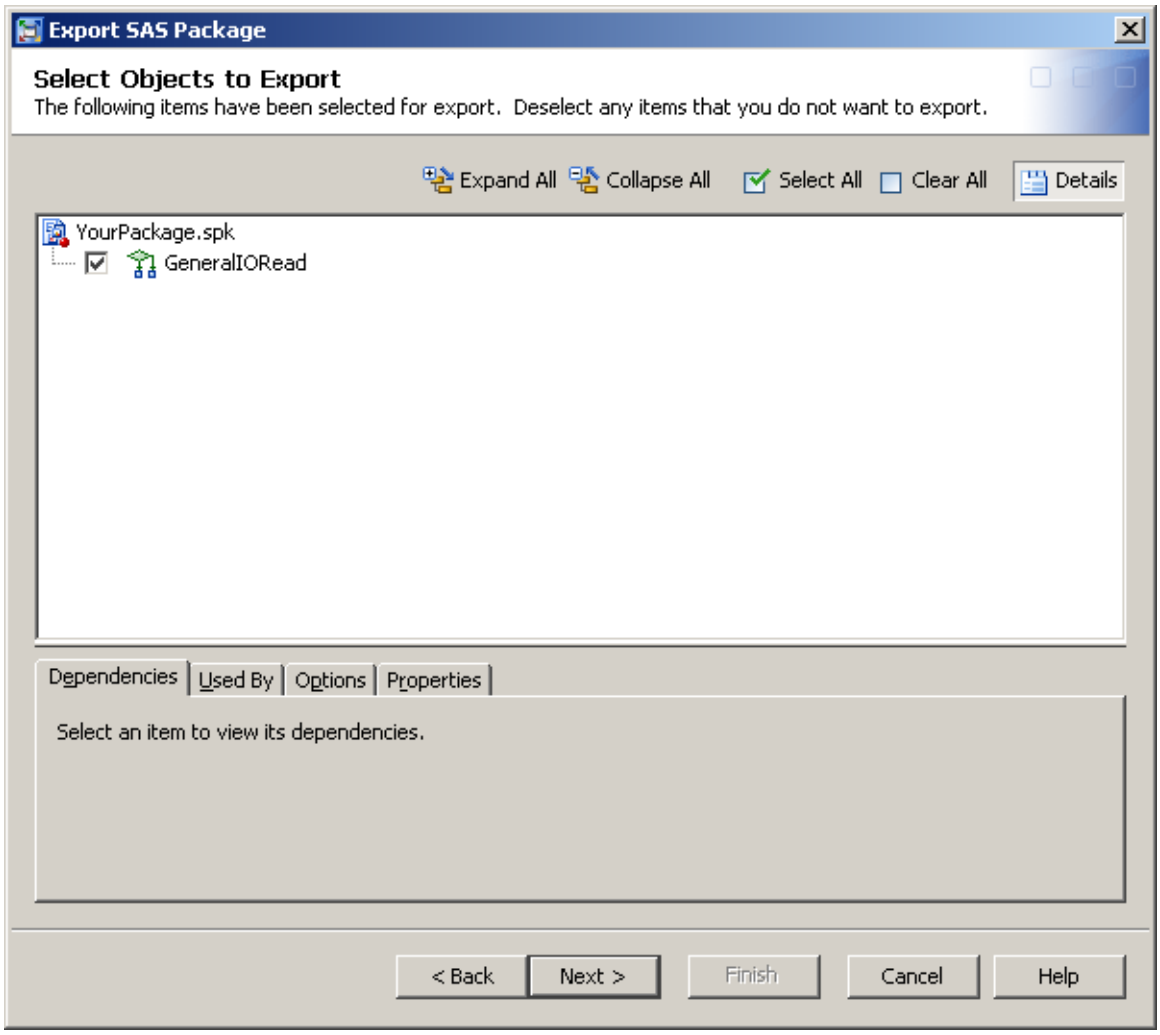
1. Open SAS Management Console and click the **Folders** tab.
2. Expand the **System** folder and the **Applications** folder.
3. Expand SAS Real-Time Decision Manager and the Real-Time Decision Manager 5.4 folders.
4. Select RTDMEngineRepository.
5. Right-click the artifact you wish to promote (for example, GeneralIORead is the artifact in the following illustration) and select **Export SAS Package** (note the previous caution).



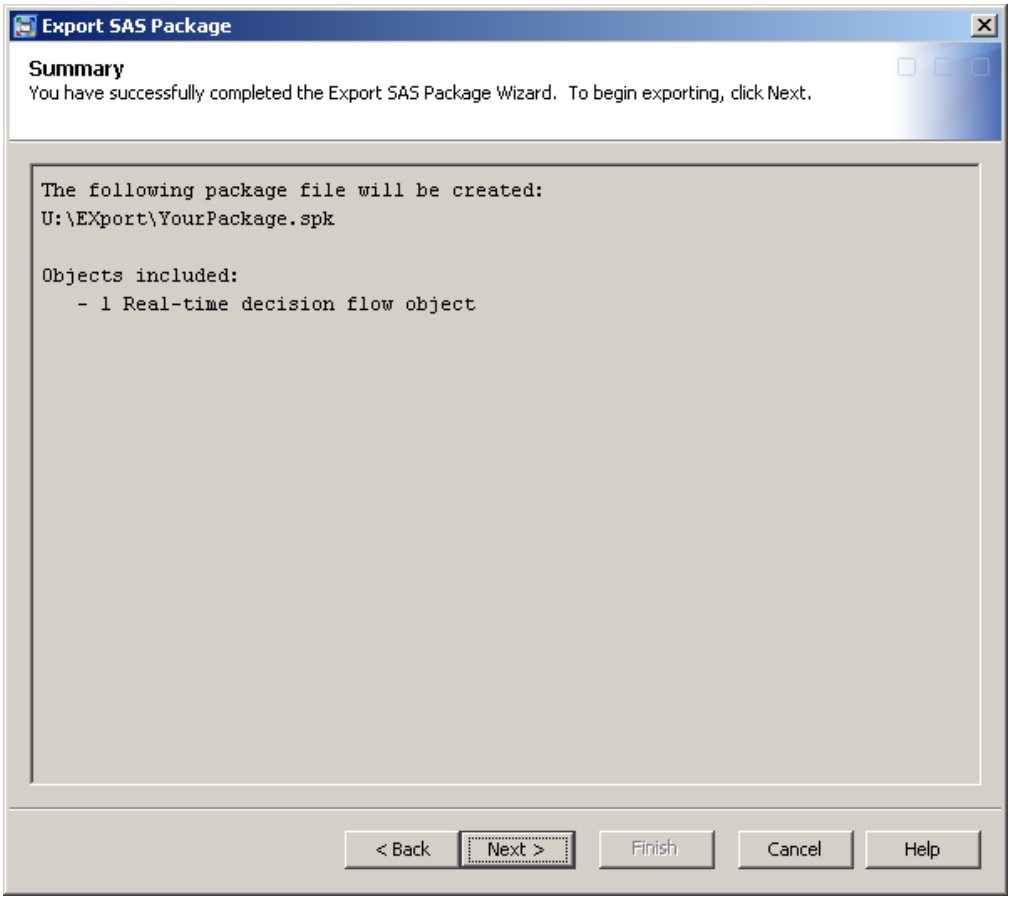

6. Enter a package name and click **Next**. When Export finishes, the package will contain the artifacts that you wish to promote.



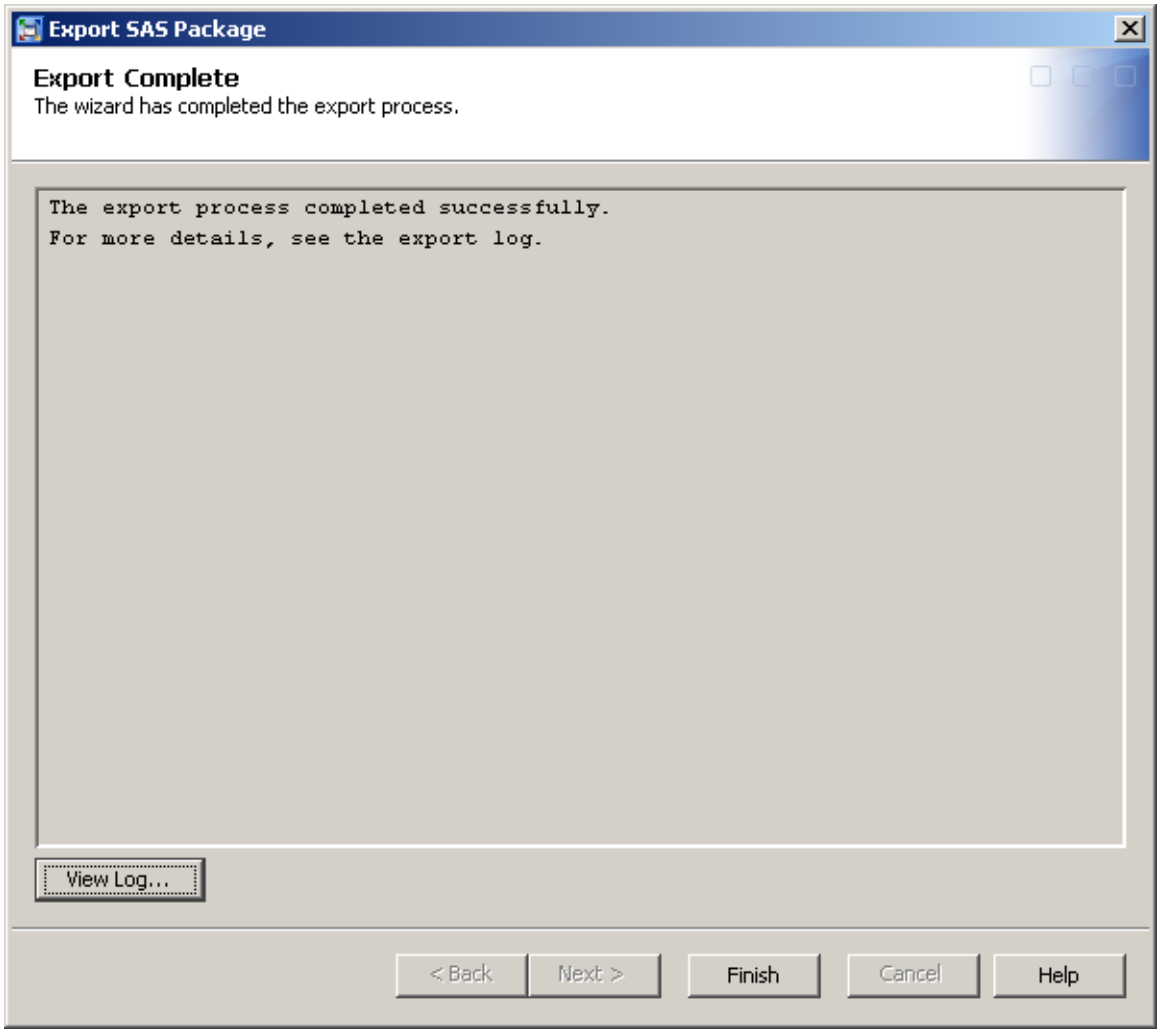

7. Check the boxes beside the artifacts that you wish to promote. If there are multiple artifacts in the folder, a convenient way to accomplish this is to click **Clear All** first, which unselects everything. Then check the box beside each xml file that you wish to promote. Click **Next**.




8. Verify the package name, location, and contents, and click **Export**.




The following dialog box appears after you successfully complete the Export function.

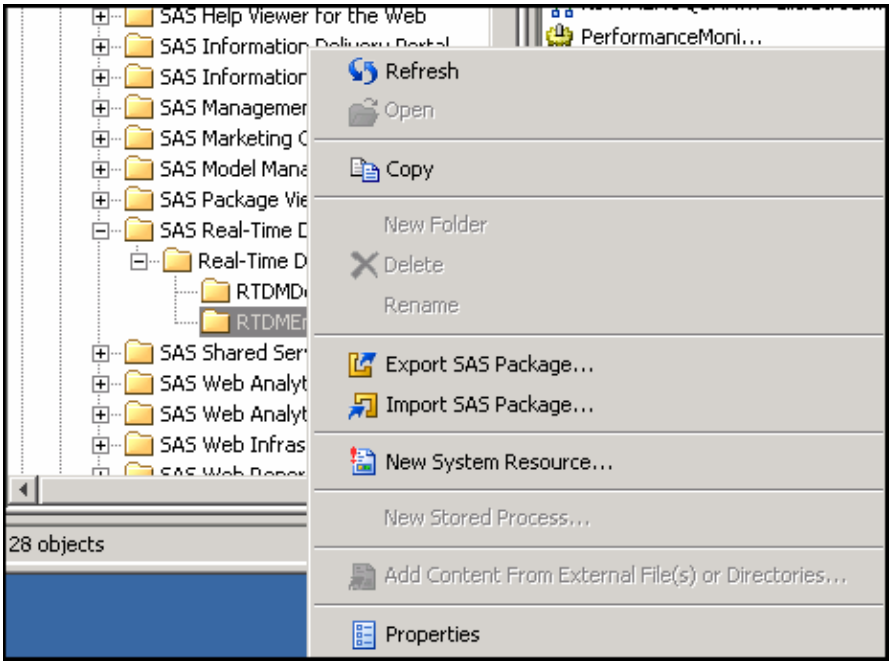



The flow has now been successfully exported from the development environment and saved in the package file called `YourPackage.spk`. The second part of the promotion process is to import the flow into the production environment.

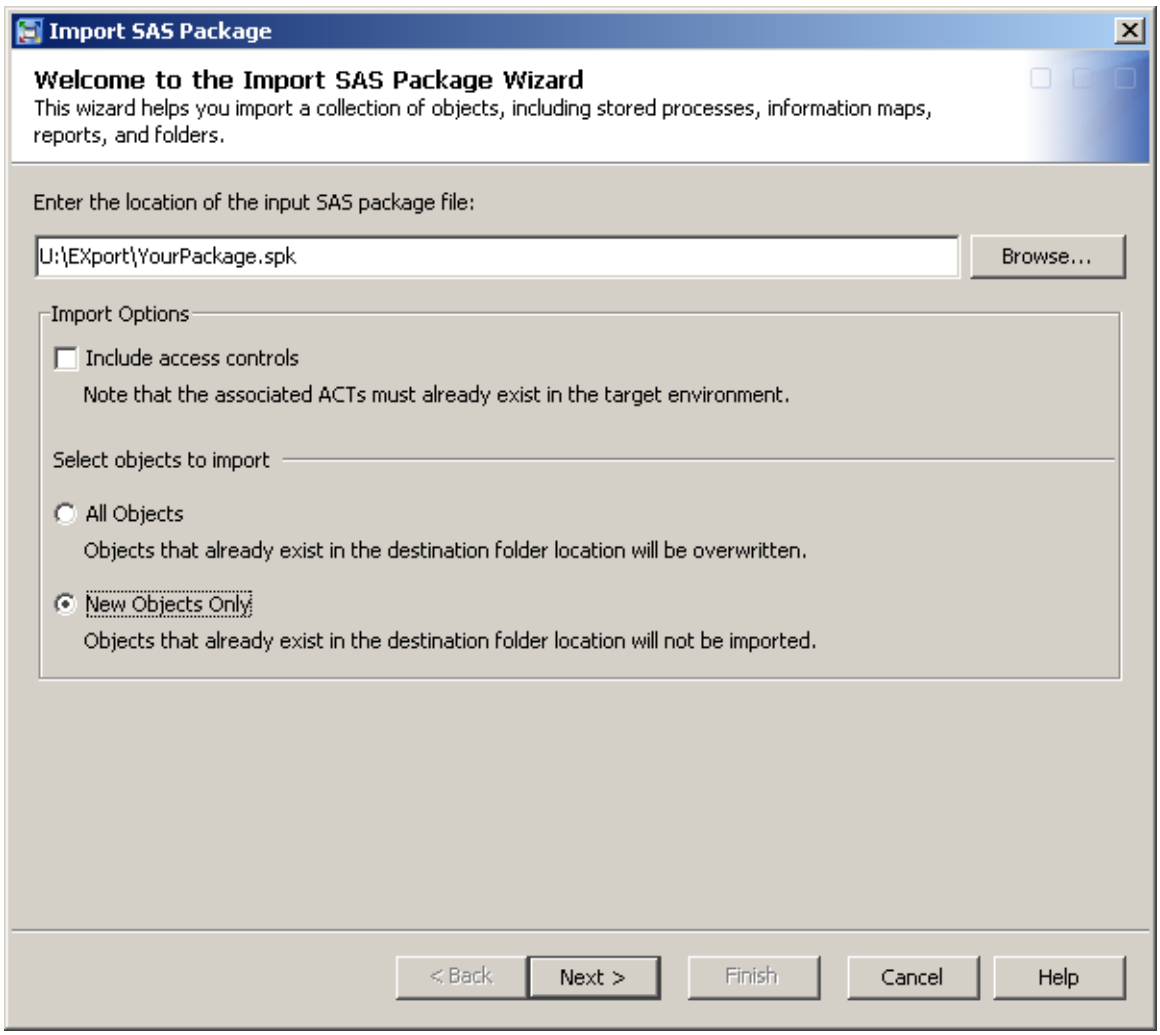
***9.*** Right-click the repository folder of the repository that you wish to promote the artifact to, and select **Import SAS Package.**

**CAUTION:** The **Folder** view in SAS Management Console does not restrict where artifacts can be imported. To avoid unpredictable results, follow these rules:

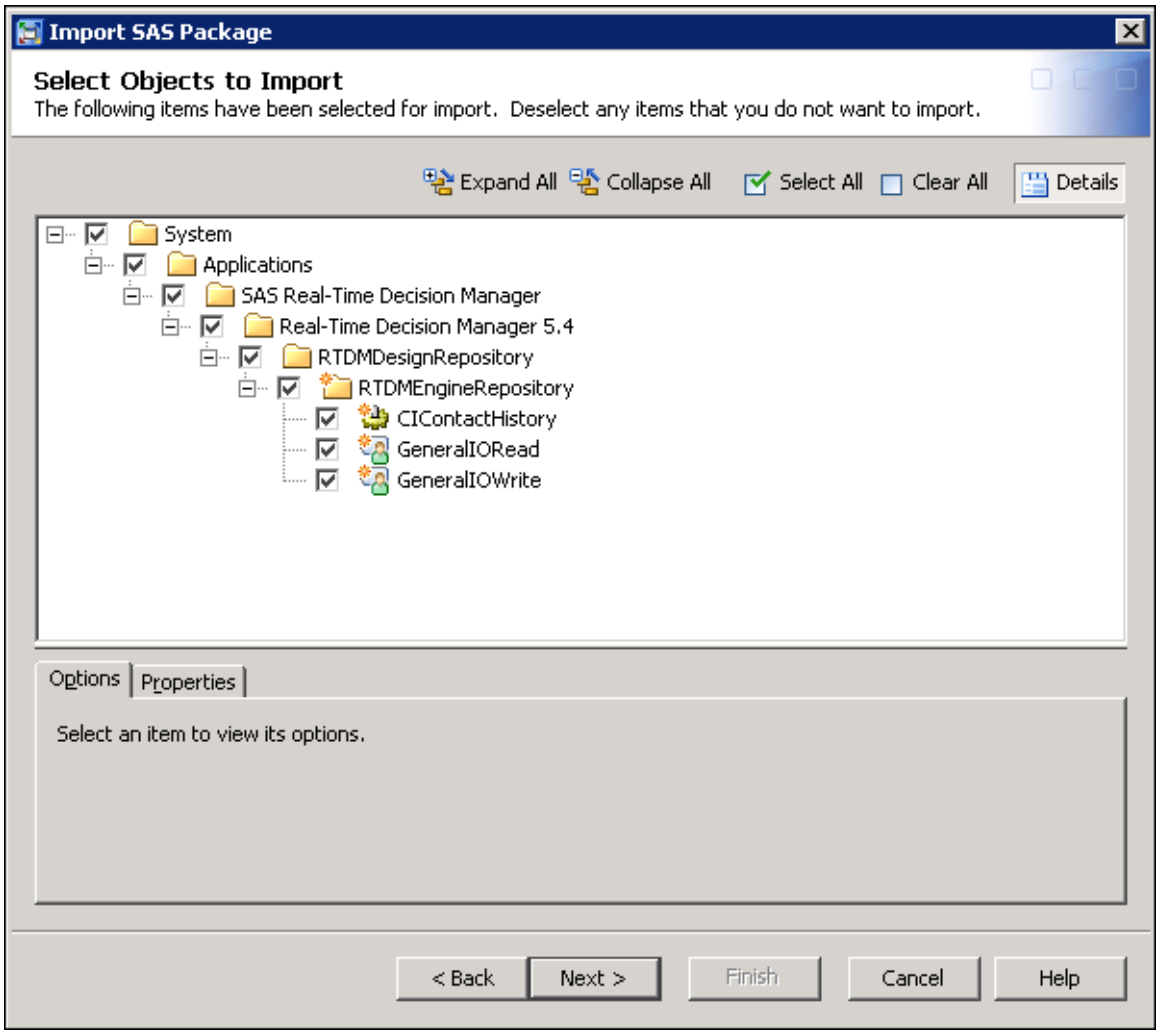
- **Always export from an individual artifact.**
- **Always import to a repository folder.**



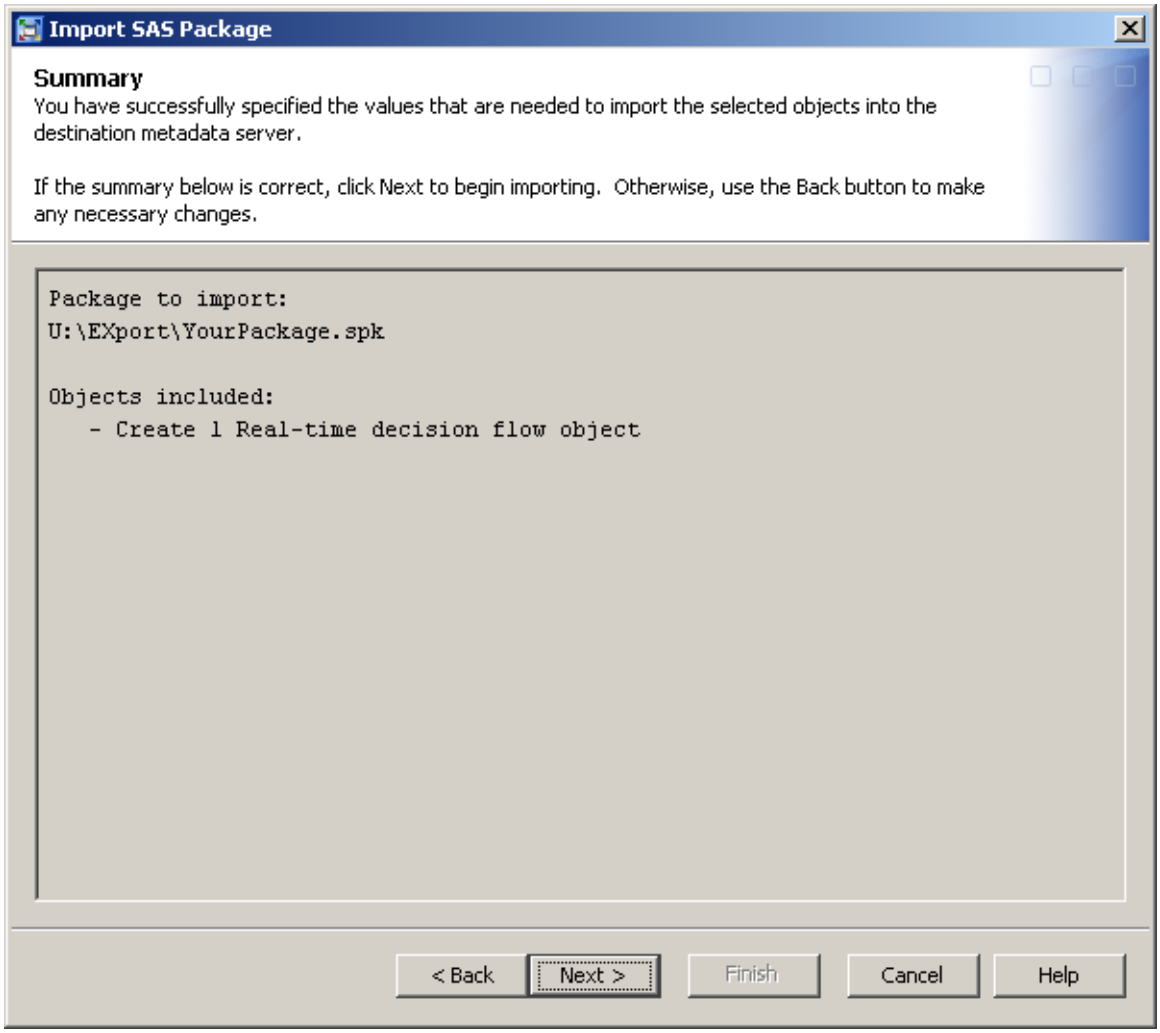

10. Browse to your package name. If you import directly after exporting, then the package name is automatically supplied. If you wish to guard against overwriting existing artifacts, then select **New Objects Only**. Click **Next**.



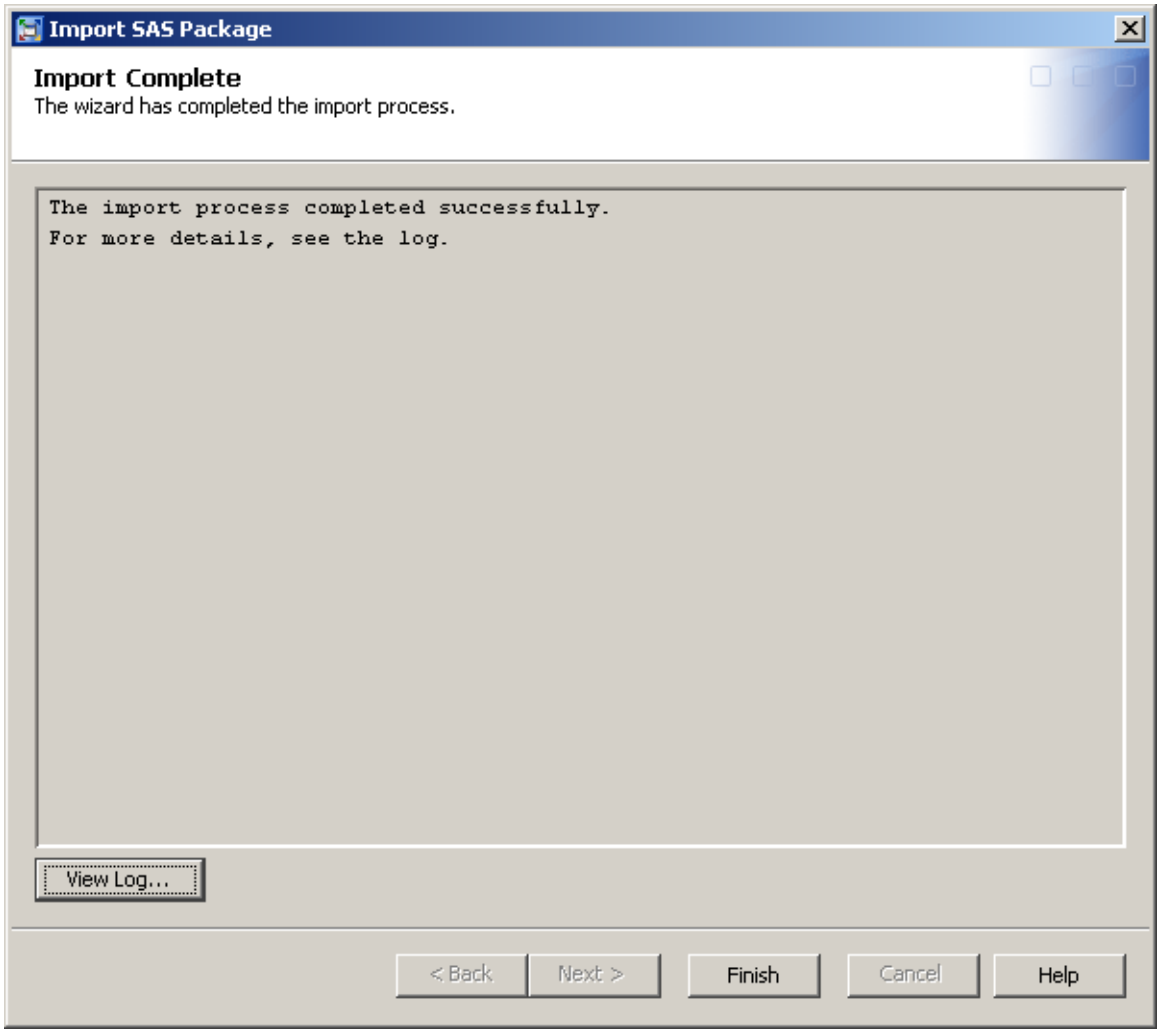

11. Verify that a check mark exists beside the xml file of each artifact that you want to import. Click **Next**.



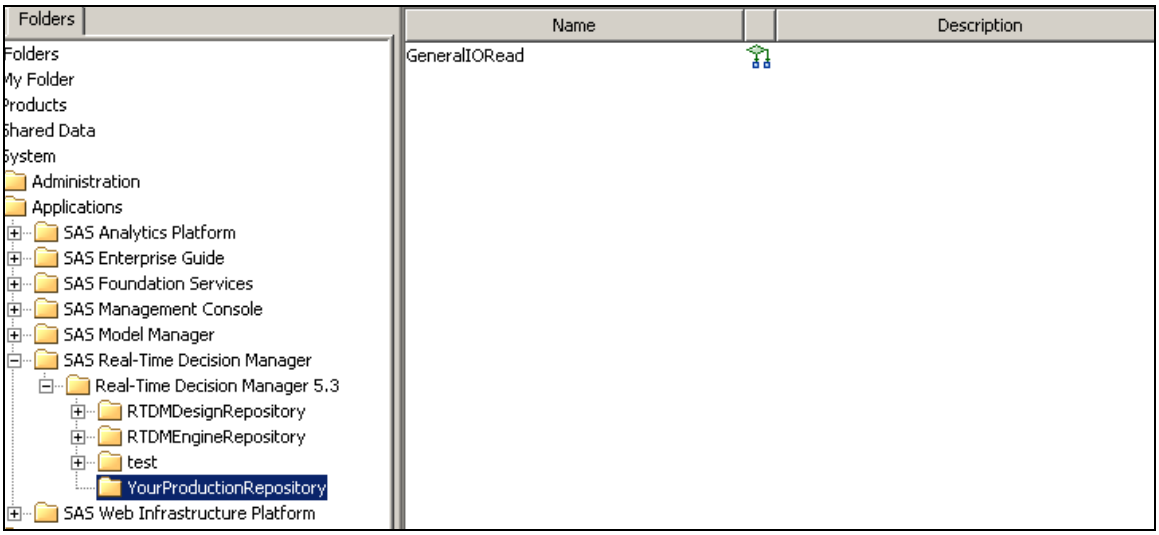

12. Verify that the summary is correct and click **Import**.




13. Click **Finish**.



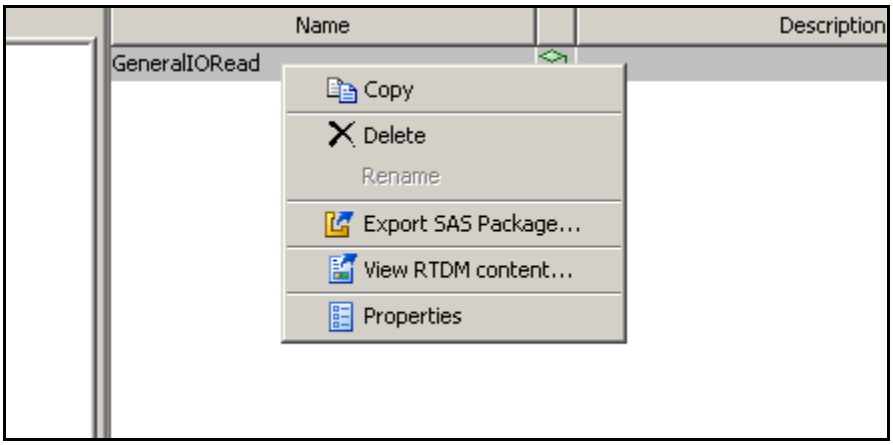

The promotion operation copies the flow without removing the flow from the source repository. The flow has been successfully promoted from the development to the production repositories in the figure below.



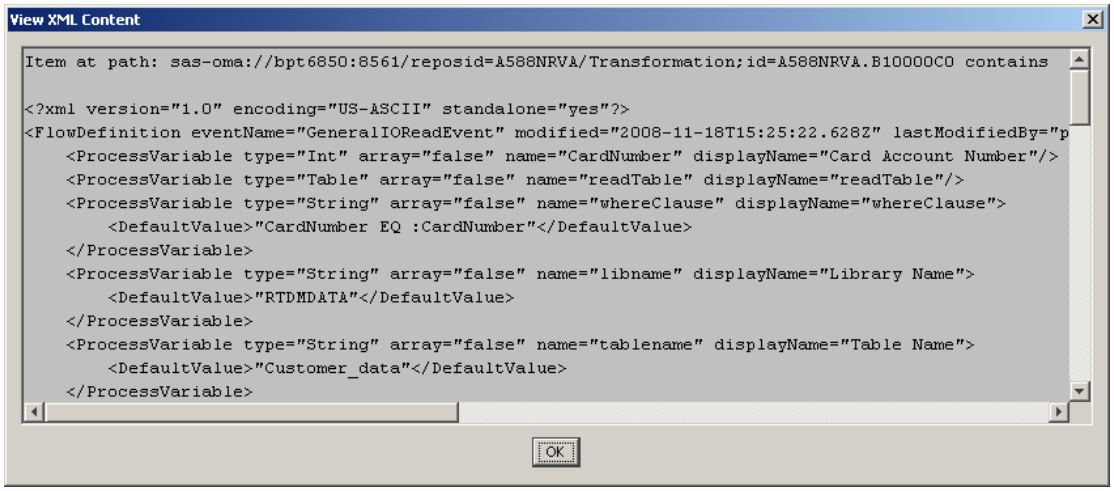

You can further verify that the promotion process was successful by viewing the contents of the xml file after promotion.

1. Click YourProductionRepository folder so that it appears in the right-hand pane.

2. Right-click GeneralIORead and select View RTDM content.




If xml content can be viewed, then the promotion was successful.




Repeat the promotion steps for each artifact type to be promoted.

# Activating Flows

When a flow is activated, the SAS Real-Time Decision Manager engine loads it, making it ready to process events. When a flow is deactivated, the engine unloads it, making it no longer ready to process events. When the engine receives an event for which there is no active flow, it returns a "no flow" fault message to the caller.

A flow is the only artifact that can be activated or deactivated. All other artifacts are used by flows, directly or indirectly, and are loaded when referenced by an active flow. When loaded, flows and other artifacts are synchronized across the machines in the SAS Real-Time Decision Manager cluster and cached in memory for maximum performance.

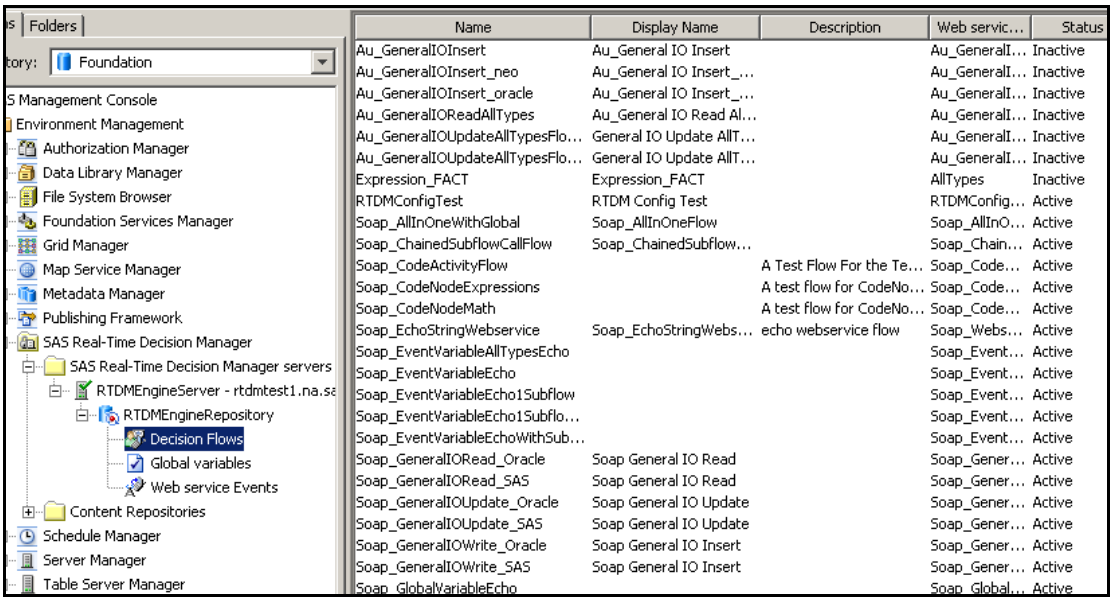
The system references flows by name and version. This allows client applications to track the updates and improvements to a flow by incrementing the version number each time an improvement is made. Therefore, when you activate, you must activate a particular version of a flow. The other artifact types are not updated in this way and are referenced by name only.

Each flow is bound to an event, which specifies the type of request a flow processes. Many different flows referencing the same event may exist in a repository, but only one of those flows can be active at any given time. To illustrate this, suppose flows *A* and *B* reference event *X* and suppose *A* is active. Now whenever event *X* is received it is routed to flow *A*. If you activate flow *B*, SAS Real-Time Decision Manager will automatically deactivate flow *A*. Now whenever event *X* is received it is routed to flow *B*.

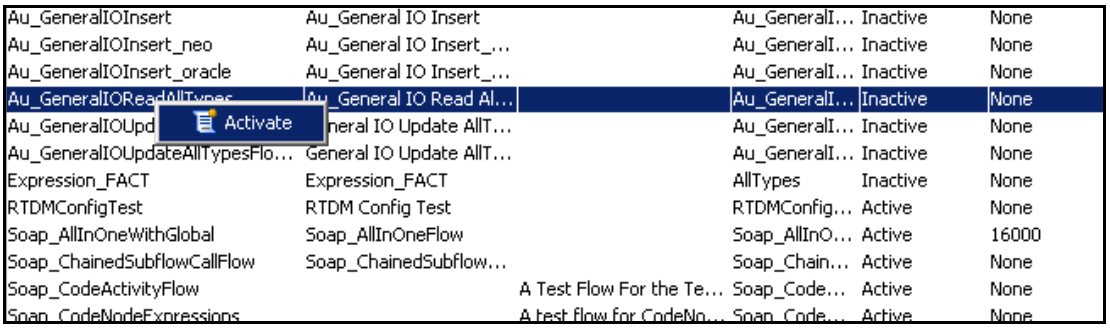
Note that it is not necessary to activate or deactivate flows in the development environment. When a flow test is run via SAS Customer Intelligence Studio, SAS Real-Time Decision Manager automatically takes care of loading, testing, and unloading the appropriate flow. Since the development environment is not connected to channels, the active/inactive states of the flows there are irrelevant.

To activate a flow:

1. Launch SAS Management Console.
2. Expand SAS Real-Time Decision Manager and SAS Real-Time Decision Manager servers.
3. Expand the SAS Real-Time Decision Manager system containing the flow that you want to activate. In the example below, RTDMEngineServer represents a running SAS Real-Time Decision Manager engine that is deployed within a cluster. The green check mark indicates that the plug-in has successfully connected to the engine.
4. Expand the repository (RTDMEngineRepository in the following example) and click **Decision Flows**.



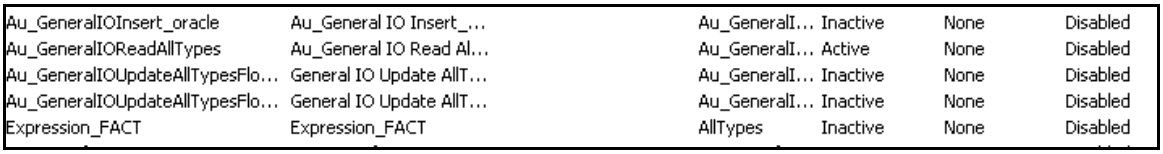

**5.** In the right-hand pane, right-click a flow and select **Activate.**




When a flow has been successfully activated, the following dialog boxes will appear:

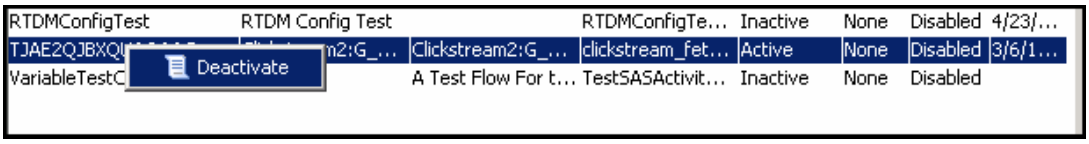


The flow status will change from inactive to active (see the following display).



| | | | | | |
|---|---|---|---|---|---|
| Au_GeneralIOInsert_oracle | Au_General IO Insert_... | Au_GeneralI... | Inactive | None | Disabled |
| Au_GeneralIOReadAllTypes | Au_General IO Read Al... | Au_GeneralI... | Active | None | Disabled |
| Au_GeneralIOUpdateAllTypesFlo... | General IO Update AllT... | Au_GeneralI... | Inactive | None | Disabled |
| Au_GeneralIOUpdateAllTypesFlo... | General IO Update AllT... | Au_GeneralI... | Inactive | None | Disabled |
| Expression_FACT | Expression_FACT | AllTypes | Inactive | None | Disabled |

To deactivate a flow, follow the previous steps in order to view the list of flows. Then right-click an active flow, and select **Deactivate** (see the following display):



# Managing Global Variables

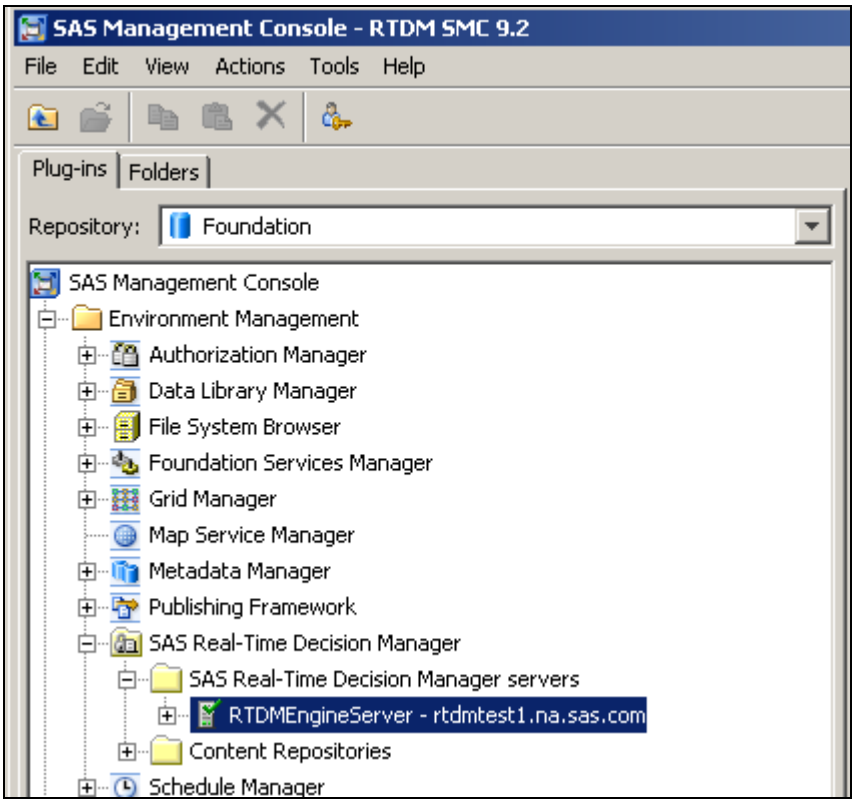
Global variables are threshold values that are used to tune the behavior of flows at execution time. Unlike process variables that are specific to a flow, the value of a global variable affects the behavior of every flow that references it. Although only an administrator can change the value of a global variable, the global variable is actually a business construct.

To illustrate this idea, suppose a financial services institution wishes to offer premium rates on short-term investment products when more than $10,000 is invested. A global variable called `MinimumInvestment` with an initial value of 10000.00 might be used in all flows that control the offers of short-term investments. Suppose it is later discovered that money is lost on such investment products when the investment is less than $12,000. Because a global variable was used, its value can easily be adjusted to 12000.00, rather than modifying every flow that controls the offering of a short-term investment.

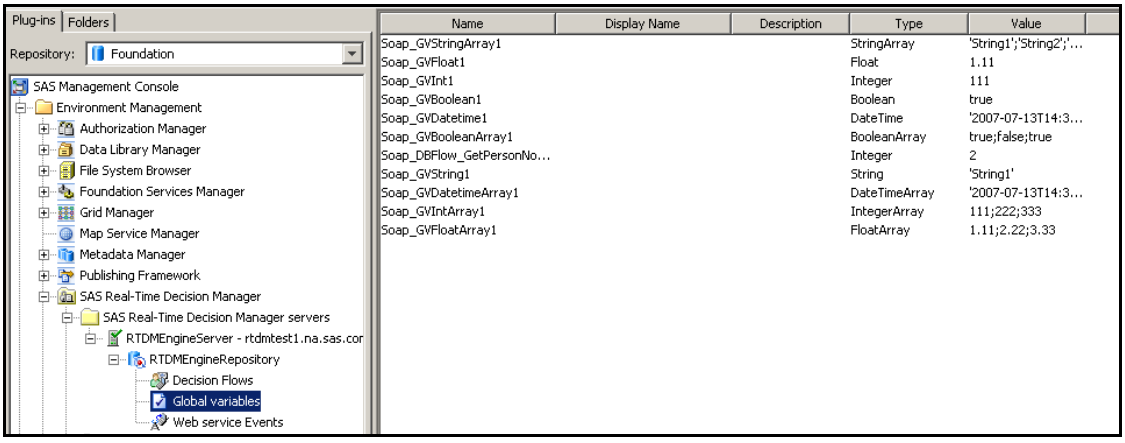
Global variables are created and assigned initial values by SAS Customer Intelligence Studio users. For security reasons, end-users can only update the contents of a development repository. Only an administrator can change the value of a global variable in a production environment.

To change the value of a global variable, follow these steps:

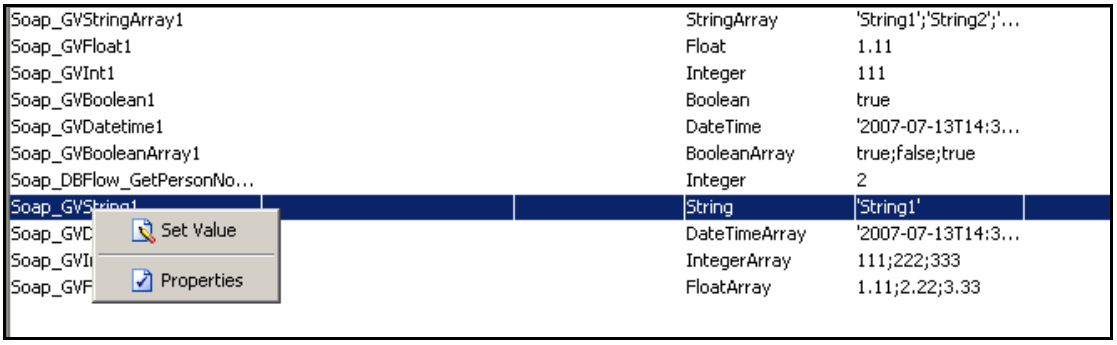
1. Open SAS Management Console.
2. In the Plug-ins folder, expand SAS Real-Time Decision Manager and SAS Real-Time Decision Manager servers.



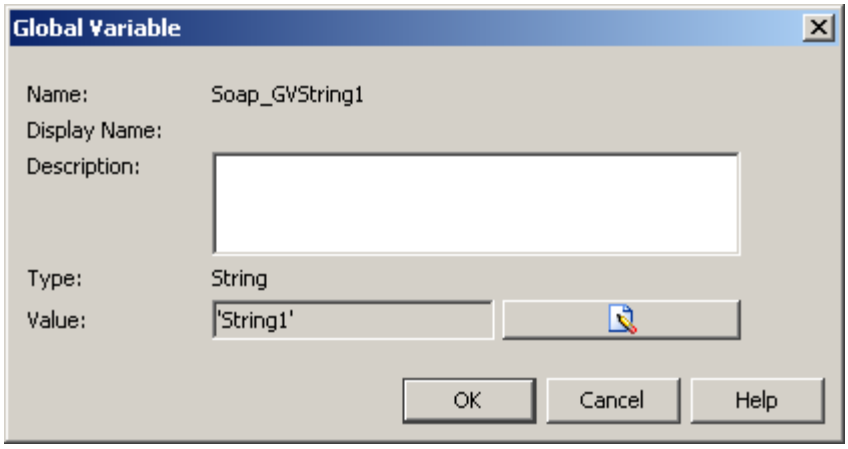

3. Expand the system that contains the global variable you wish to update. Expand the repository, and the **Global Variables** folder.



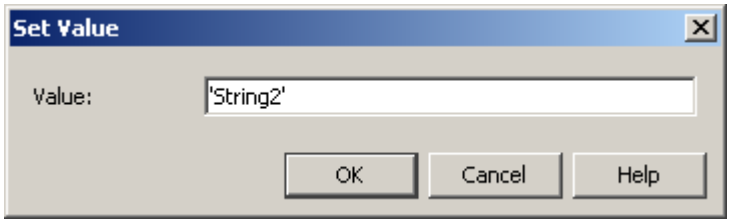

4. Right-click the global variable that you want to change and select **Properties.**




5. Click the **Edit** button to edit the value.




6. Type in the new value and click **OK.** Use either single or double quotation marks to indicate a string value.




The new value is displayed in the table on the right pane.

| | | |
|---|---|---|
| Soap_GVStringArray1 | StringArray | 'String1';'String2';'... |
| Soap_GVFloat1 | Float | 1.11 |
| Soap_GVInt1 | Integer | 111 |
| Soap_GVBoolean1 | Boolean | true |
| Soap_GVDatetime1 | DateTime | '2007-07-13T14:3... |
| Soap_GVBooleanArray1 | BooleanArray | true;false;true |
| Soap_DBFlow_GetPersonNo... | Integer | 2 |
| Soap_GVString1 | String | 'String2' |
| Soap_GVDatetimeArray1 | DateTimeArray | '2007-07-13T14:3... |
| Soap_GVIntArray1 | IntegerArray | 111;222;333 |
| Soap_GVFloatArray1 | FloatArray | 1.11;2.22;3.33 |

# Set an Event Time-Out

The SAS Real-Time Decision Manager engine receives, processes and responds to requests submitted by the client. When defining an event in the SAS Management Console, an administrator will be able to specify a time-out setting for the event. A decision campaign or diagram associated with the event will have different maximum time to complete a requested transaction, even for the same channel. This capability ensures that a decision campaign or diagram will provide a response within a specified time that is appropriate for the channel and the type of customer interaction. If the run-time engine does not respond to the event request by the time-out interval, then a fault is raised.

There are three levels at which time-out values can be set: system, event, and flow.

The lowest level that a time-out can be set is at the system level. The system level can be set in the SAS Real-Time Decision Manager EAR during installation and configuration of the design server and engine. If no value is specified by the user, then a default value is set.

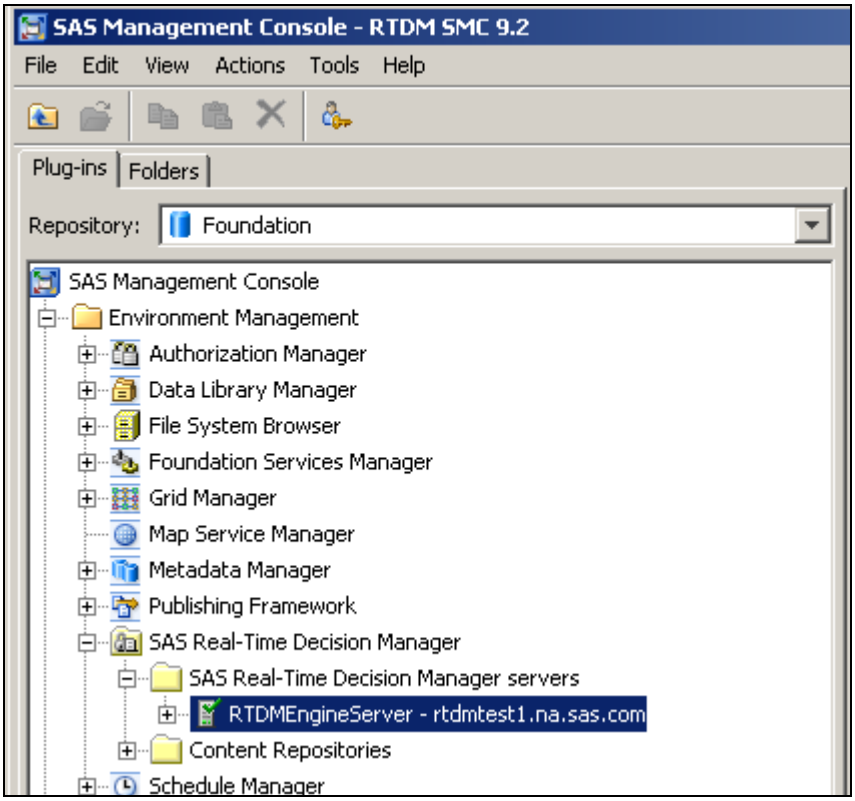
The next level that a time-out value can be set is at the event level. Use the SAS Real-Time Decision Manager plug-in of the SAS Management Console to set the time-out value at the event level.

The highest level of time-out is set at the flow level. The flow time-out value supersedes the event and system time-out values. Use the SAS Customer Intelligence Plug-in for SAS Management Console to set the time-out at the flow level. Specify the time-out value interval in milliseconds.

If a sub-flow is called, then the time-out value of the top-level flow or event is used. If the time-out values of the flow or the event are not specified, then the system time-out value is used.

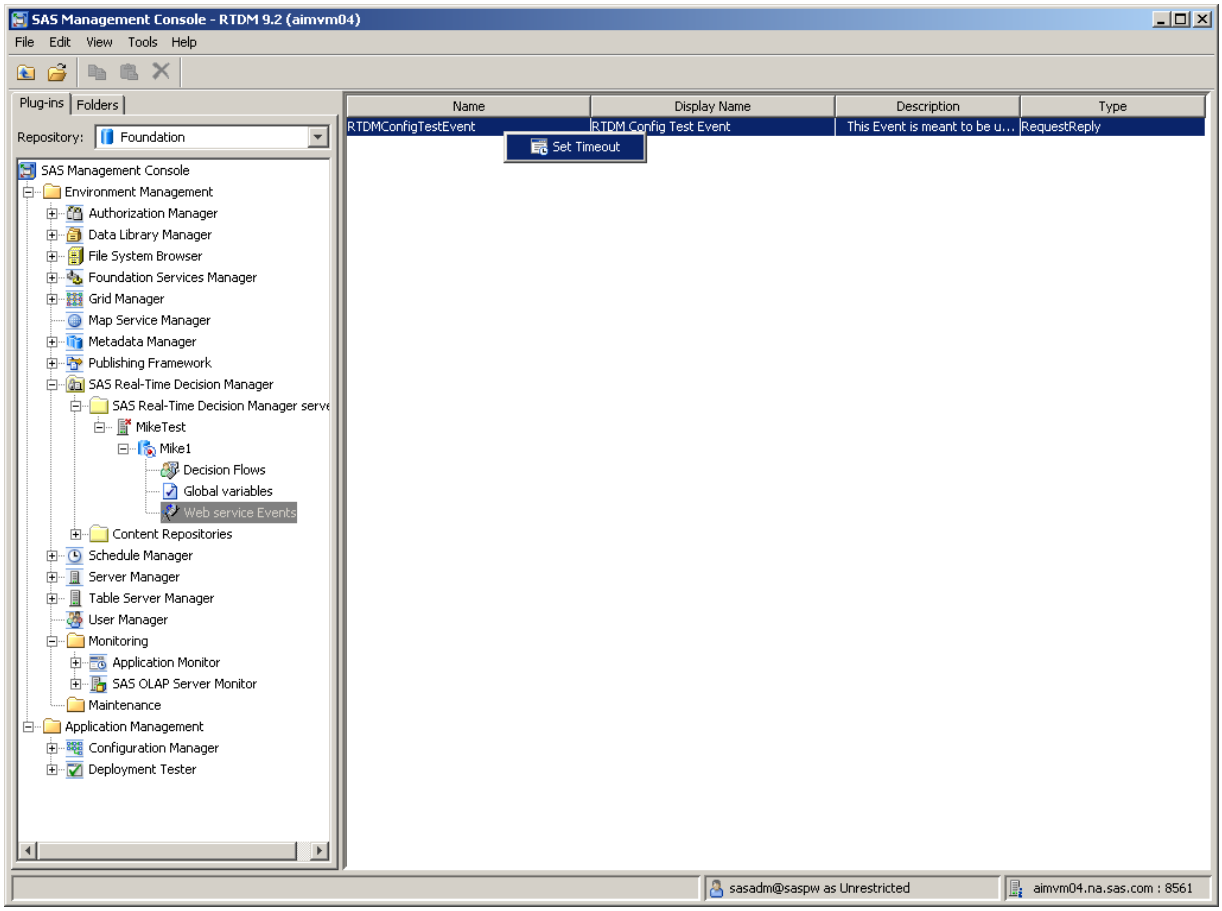
Set the event time-out value by using the SAS Real-Time Decision Manager plug-in of the SAS Management Console. To set the time-out value for an event, follow these steps:

1. Open SAS Management Console.
2. Expand SAS Real-Time Decision Manager and SAS Real-Time Decision Manager servers.


SAS Management Console - RTDM SMC 9.2
File Edit View Actions Tools Help
Plug-ins Folders
Repository: Foundation
SAS Management Console
Environment Management
Authorization Manager
Data Library Manager
File System Browser
Foundation Services Manager
Grid Manager
Map Service Manager
Metadata Manager
Publishing Framework
SAS Real-Time Decision Manager
SAS Real-Time Decision Manager servers
RTDMEngineServer - rtdmtest1.na.sas.com
Content Repositories
Schedule Manager

Expand the system that contains the Web service event that you wish to update. Expand the repository and the **Web Service Events** folder.

3. Right-click the Web service event that you want to change, and select **Set Timeout.**




4. Check **Enable** to edit the time-out value, type the value, and click **OK**. If **Enable** is cleared, then the time-out value for the event is disabled.

# Repository Management

Before using any of the advanced SAS Real-Time Decision Manager functions such as creating a new repository, make sure that you understand how to administer content repositories, or contact your on-site SAS support personnel for assistance.

When your product was installed and configured by on-site SAS support personnel, SAS Real-Time Decision Manager repositories were created and configured for you. These repositories should be sufficient to meet the requirements of your organization at the time your system was installed.

The SAS Real-Time Decision Manager plug-in to SAS Management Console provides advanced functions that allow you to create additional repositories and to delete existing repositories that are no longer needed.

## Repository Creation

Repositories are used to contain decision flows and their building blocks. These building blocks include events, activities, global variables, and system resources. You specify a repository as a development, testing, or production repository.

A development repository is the only repository type that a SAS Real-Time Decision Manager *design* server can store artifacts into. (A SAS Real-Time Decision Manager *engine* server may read data from any repository type.) However, a production environment should never be configured to utilize a development repository. Any engine that is connected to a development repository should only be used for testing decision flows.

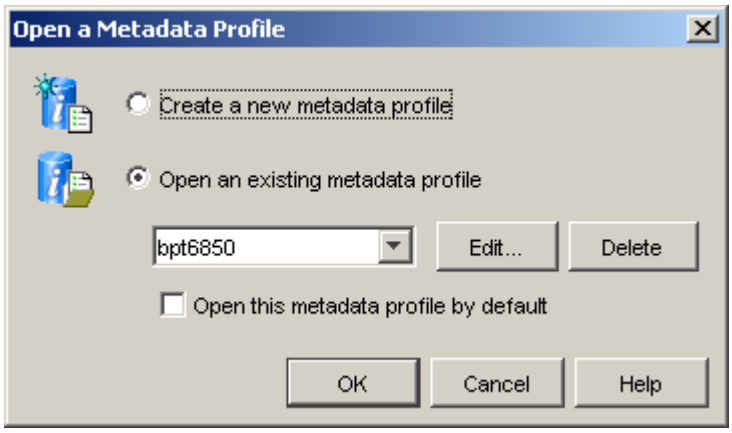
A repository does not have to be associated with a SAS Real-Time Decision Manager server; it can be used simply as a storage area for artifacts.

A repository resides in SAS Open Metadata Repository (OMR).

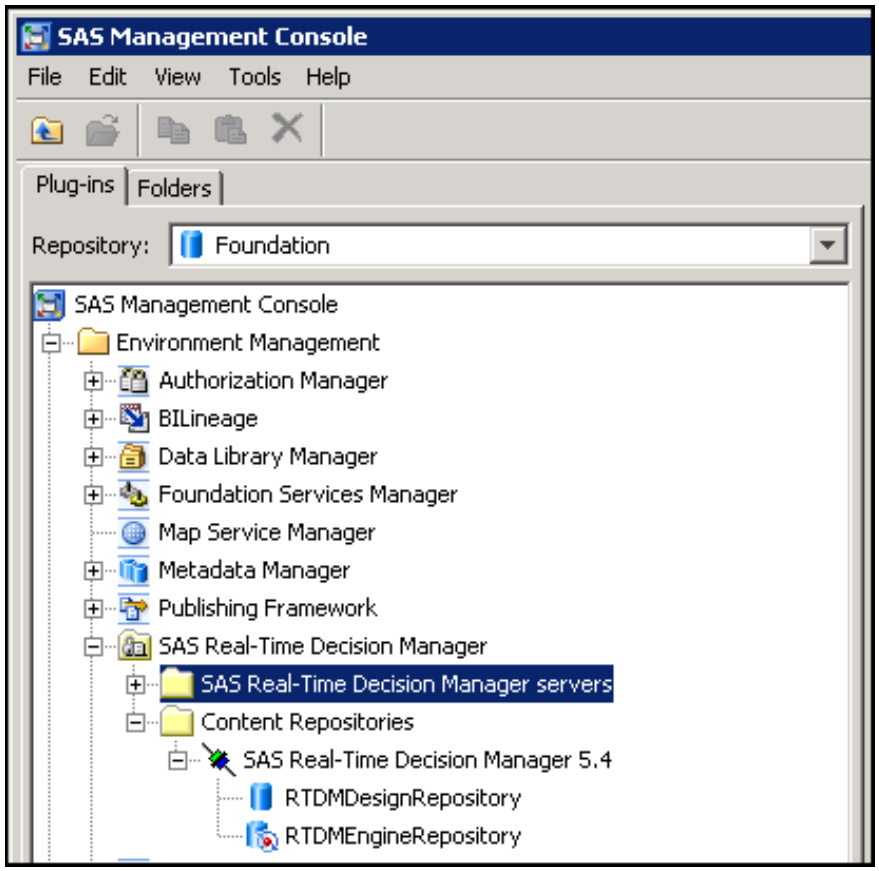
First you must decide where to locate your new repository. That is, you must decide which OMR and DAV to use. The OMR is chosen by selecting the appropriate metadata profile when you log in to SAS Management Console. See SAS Management Console User's Guide for more information about metadata profiles.

To create a new SAS Real-Time Decision Manager repository, follow these steps:

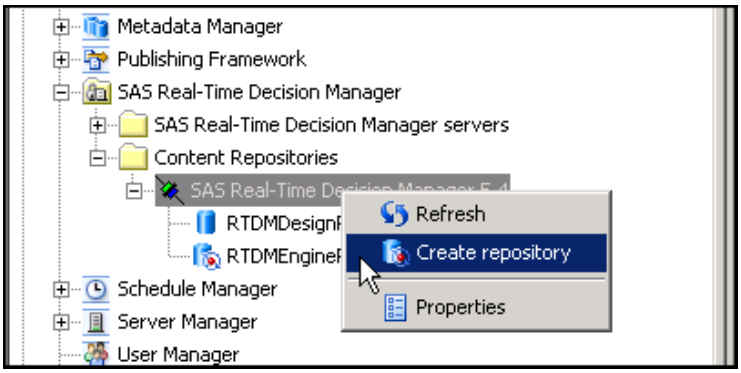
1. Log in to SAS Management Console. Select the metadata profile that is associated with the OMR where you wish to create your repository.



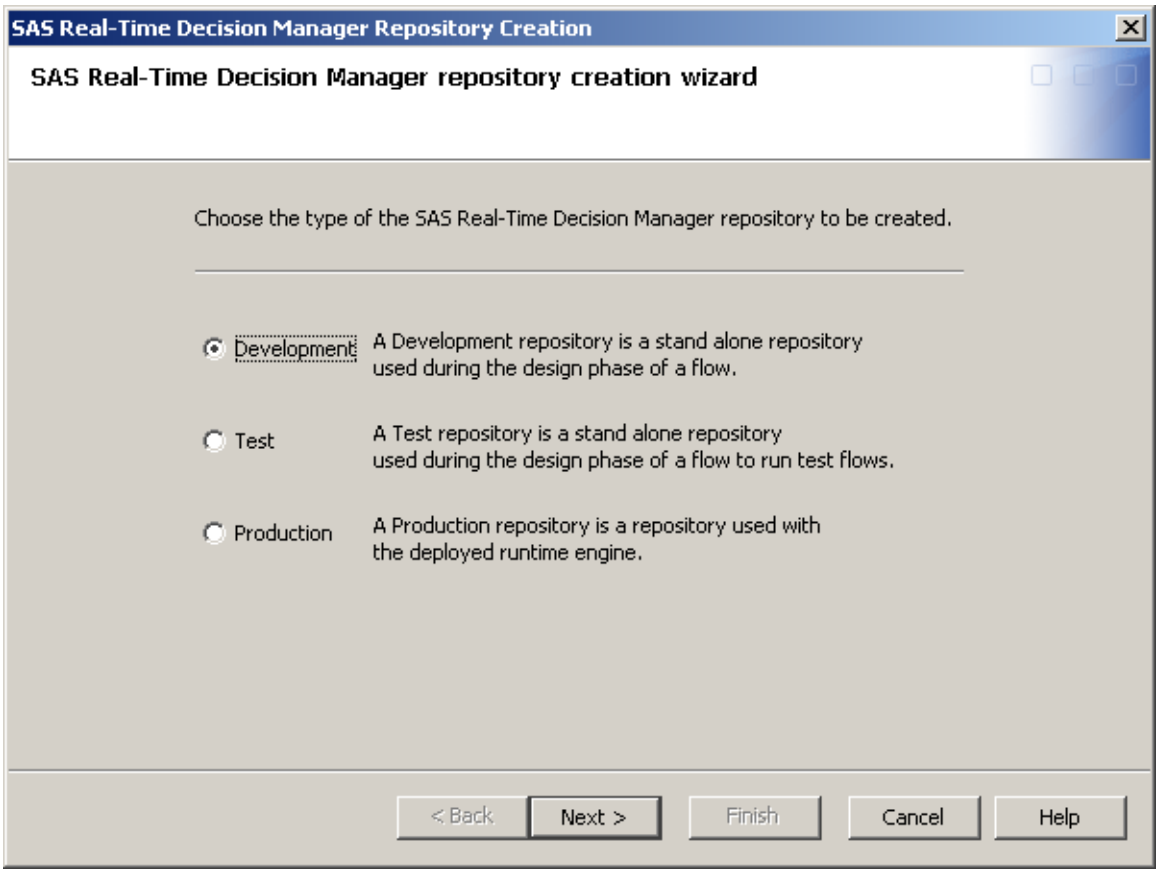

2. Expand SAS Real-Time Decision Manager and Content Repositories.



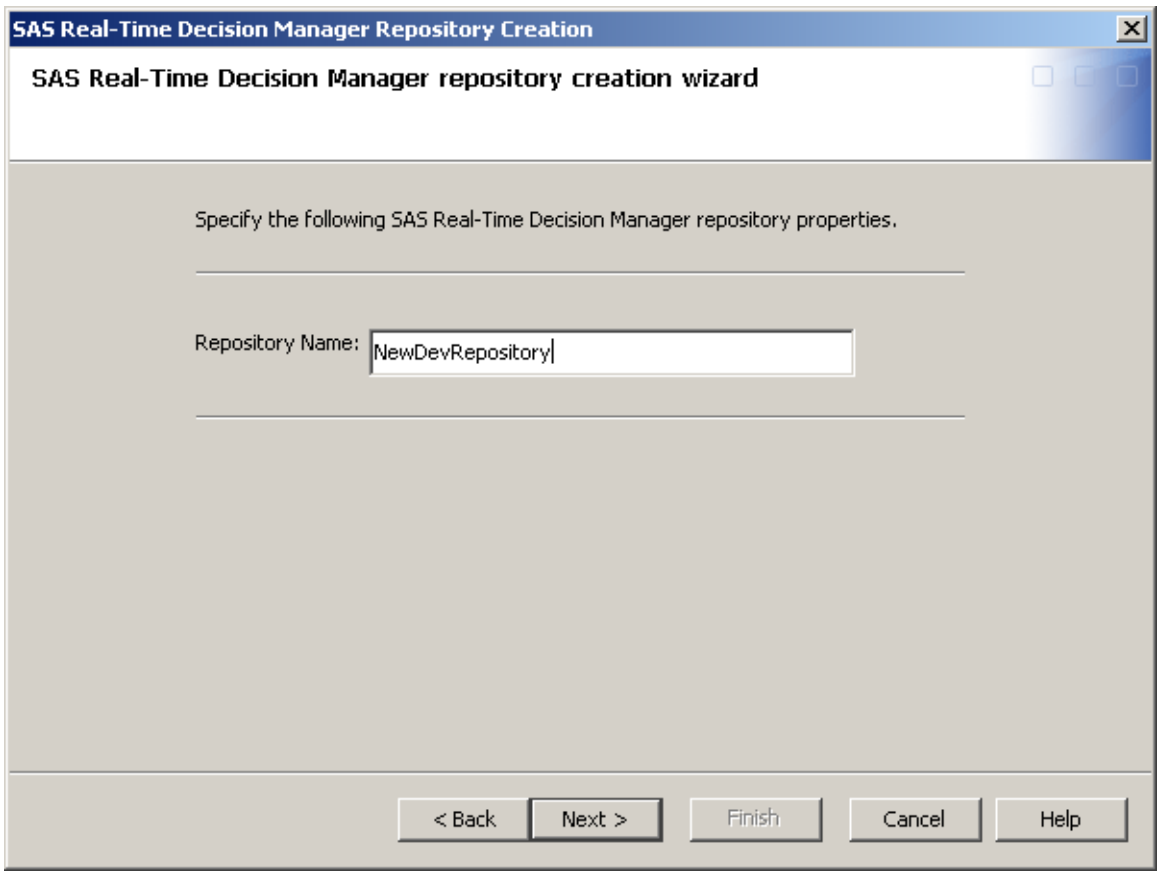

3. Right-click the content mapped folder where you wish to create your repository, and select **Create repository**.



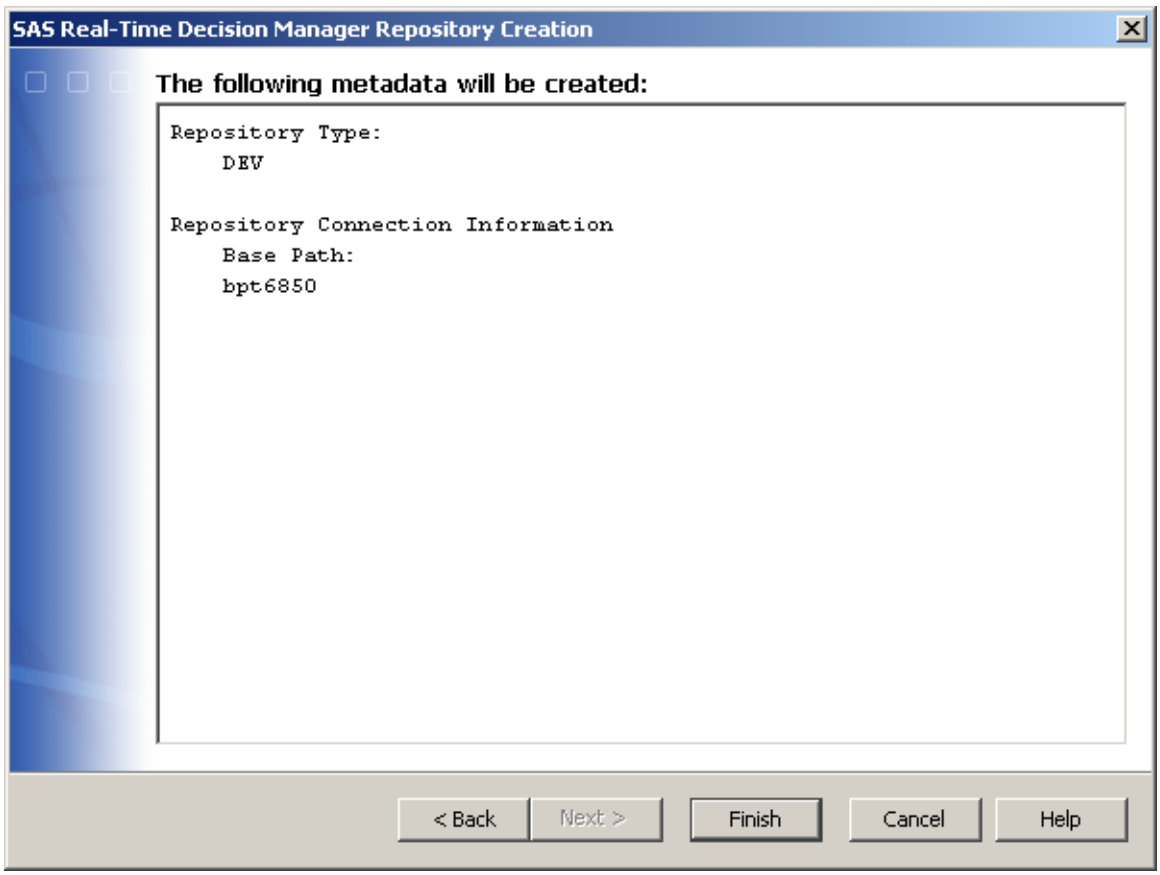

4. Choose a development, test, or production repository. Click **Next.**



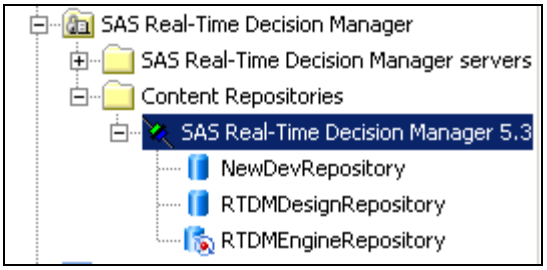

5. Enter a name for your new repository. Here, we are creating a new repository called `NewDevRepository`. Click **Next**.




6. Review the information for accuracy. Click **Finish**.




7. Verify that your repository was created correctly. Expand your repository folder. If your repository was created successfully you should see the following type folders.



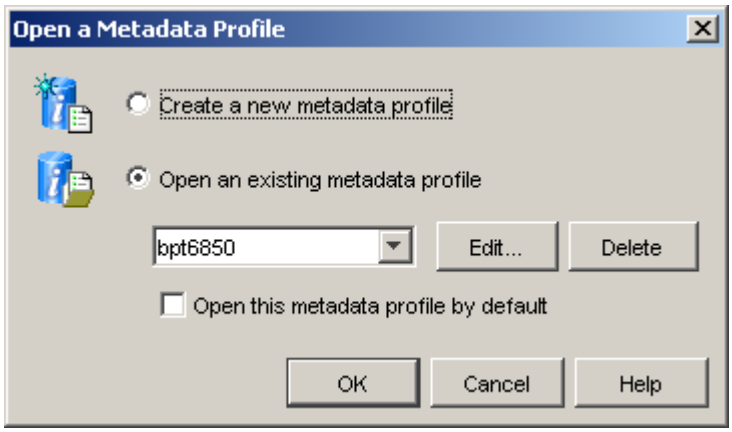

A repository is bound to a SAS Real-Time Decision Manager engine or design server when that server is installed and configured. See Installation and Configuration for details.

## Repository Deletion

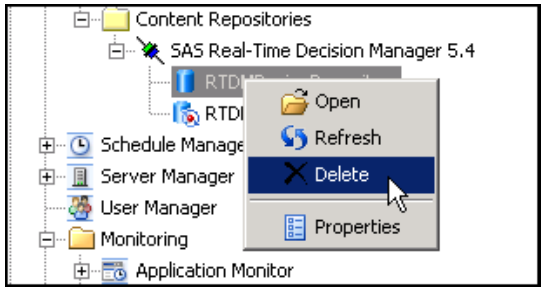
**CAUTION:** Deleting a repository is an irreversible operation.

To delete a repository:

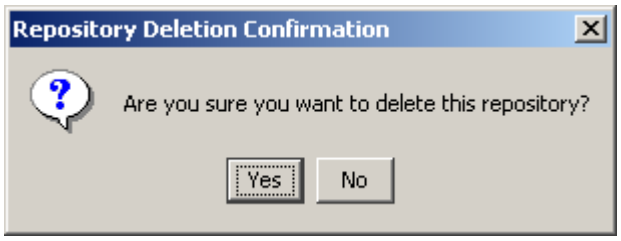
1. Log in to SAS Management Console. Choose the metadata profile that is associated with the OMR that contains the repository to delete.




2. Expand **SAS Real-Time Decision Manager** and **Content Repositories**. Right-click the repository that you want to delete and select **Delete.**




3. Verify your intent to delete the repository by clicking **Yes**.



# System Resources

System resources provide the mechanism by which the SAS Real-Time Decision Manager engine server accesses and interacts with resources outside of the J2EE middle tier such as SAS servers. Activities reference the system resources by name.

For example, many activities rely on running a SAS program to produce results. The middle tier portion of these activities must communicate with SAS back-end servers. A system resource named "SAS Connection" provides the information needed to facilitate such communications. More specifically, the SAS Connection system resource contains information needed by a SAS activity to post a request queue and listen to a reply queue. On the SAS back-end server, a Message Queue Polling server process listens to the same request queue and posts responses to the same reply queue. This allows the middle tier portion of a SAS activity to post requests for processing to the SAS tier.

The fact that activities reference system resources by name, rather than contain the resource information directly, allows flows to be portable between systems. The product supports configurable development, test, and production environments. Typically, the sets of back-end SAS servers that are used by development, test, and production environments are different. System resources enable the correct set of servers to be used in each environment without modification of flows or activities. That is, each environment contains system resources with the same names. The information contained by these system resources differs from environment to environment, however.

## About SAS Connection System Resources

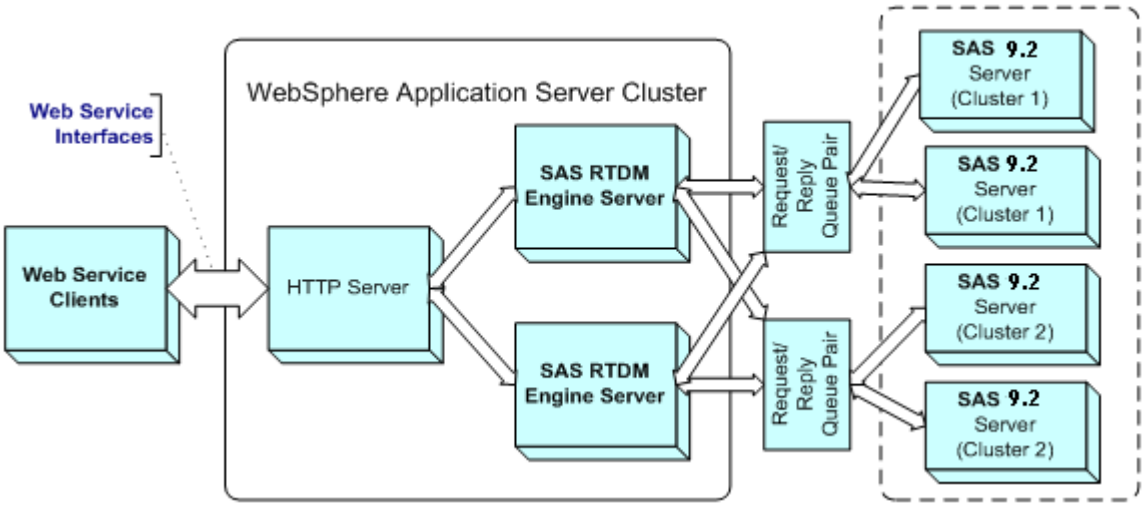
Most activities utilize SAS programs running on the SAS server tier. In order to allocate computing resources efficiently, the SAS server tier may be organized into SAS server clusters. Every server within a given cluster processes the same activity set. Let's illustrate this concept with an example.

Imagine there are two SAS server clusters, *Cluster A* with 3 servers and *Cluster B* with 2 servers. Suppose also that we have some scoring activities that are very computationally intensive. In order to allocate hardware resources efficiently, scoring activities should be assigned to *Cluster A* and all other activities to *Cluster B*. This can be accomplished with two SAS Connection system resources, one for each SAS cluster.

The following activity types use the SAS Connection system resource:

- SAS Activity
- Scoring Activity
- General I/O Activity

SAS Connection resources specify which SAS back-end cluster each SAS Activity will utilize. In SAS Real-Time Decision Manager 5.4, WebSphere MQ is the communications vehicle that provides communications between the SAS Real-Time Decision Manager Engine Server on the J2EE middle tier and the SAS Real-Time Decision Manager SAS Connection Server running on SAS back-end servers.




Each SAS server cluster has one request queue and one response queue associated with it. Activities running in the J2EE middle tier post messages to the request queue. The first available SAS server in the cluster pulls a message off the request queue, processes the message, and posts a response to the response queue. The requesting activity pulls the response from the response queue.

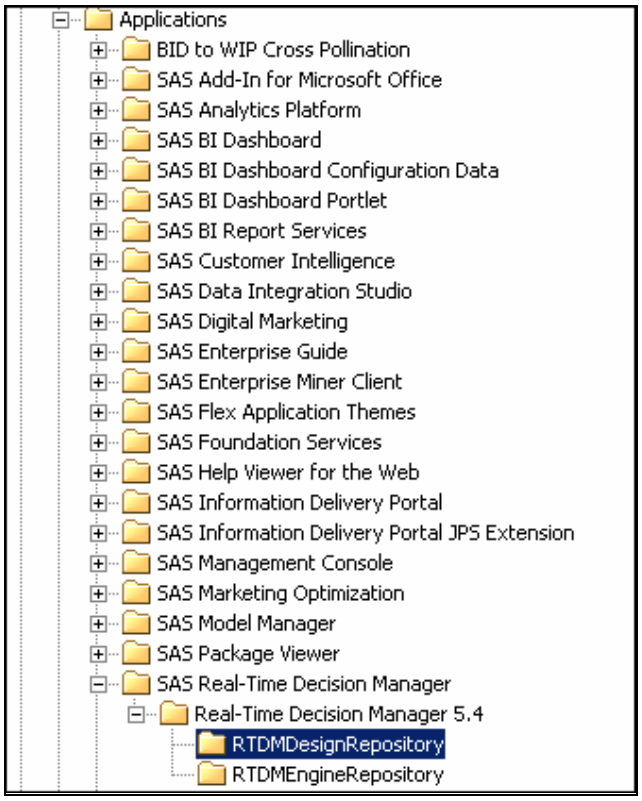
Each SAS server is associated with a queue pair during configuration of the Message Queue Polling server. Because decision flows must be portable, request/reply queue pairs are assigned to activities indirectly by associating them with SAS Connection system resources. This mechanism allows a flow, without modification, to use an entirely different set of SAS servers in the production environment than it used in the development environment.

## Specifying a New System Resource as a SAS Connection

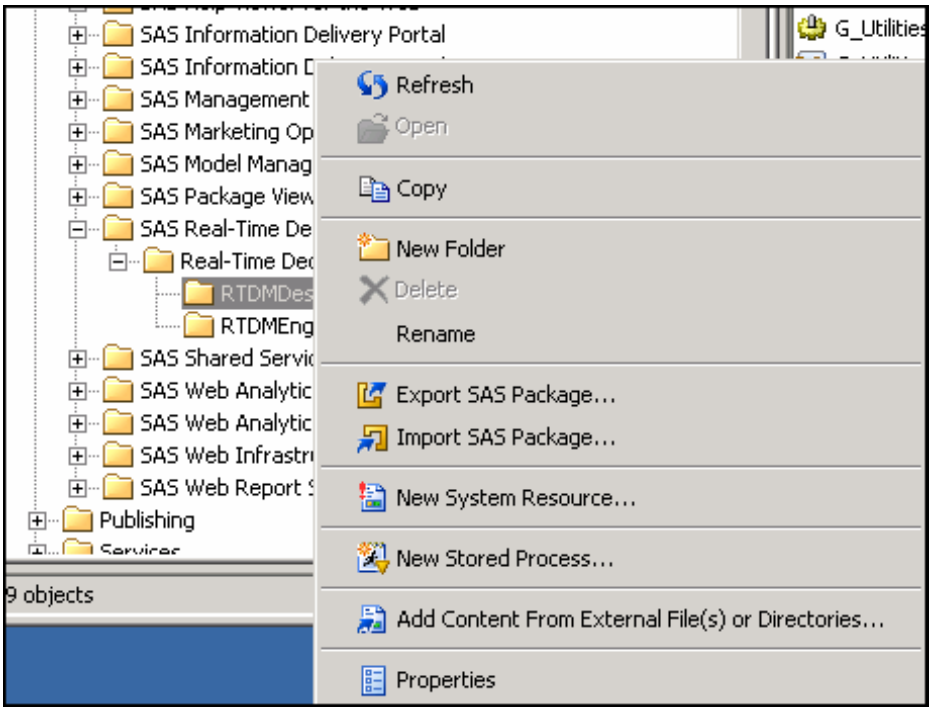
Use the following procedure to create a new SAS Connection system resource. It is assumed that WebSphere MQ is installed and configured, and that the appropriate queue manager and queues have been created.

To create a new system resource as a SAS Connection, select the **Folders** tab, and follow these steps:

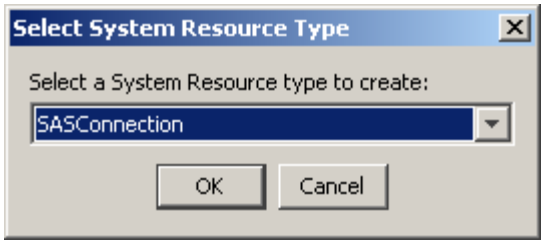
1. On the Folders tab, expand System ➧ Applications ➧ SAS Real-Time Decision Manager ➧ Real-Time Decision Manager 5.4.
2. Right-click a repository folder (**RTDMDesignRepository** is selected in the following example).



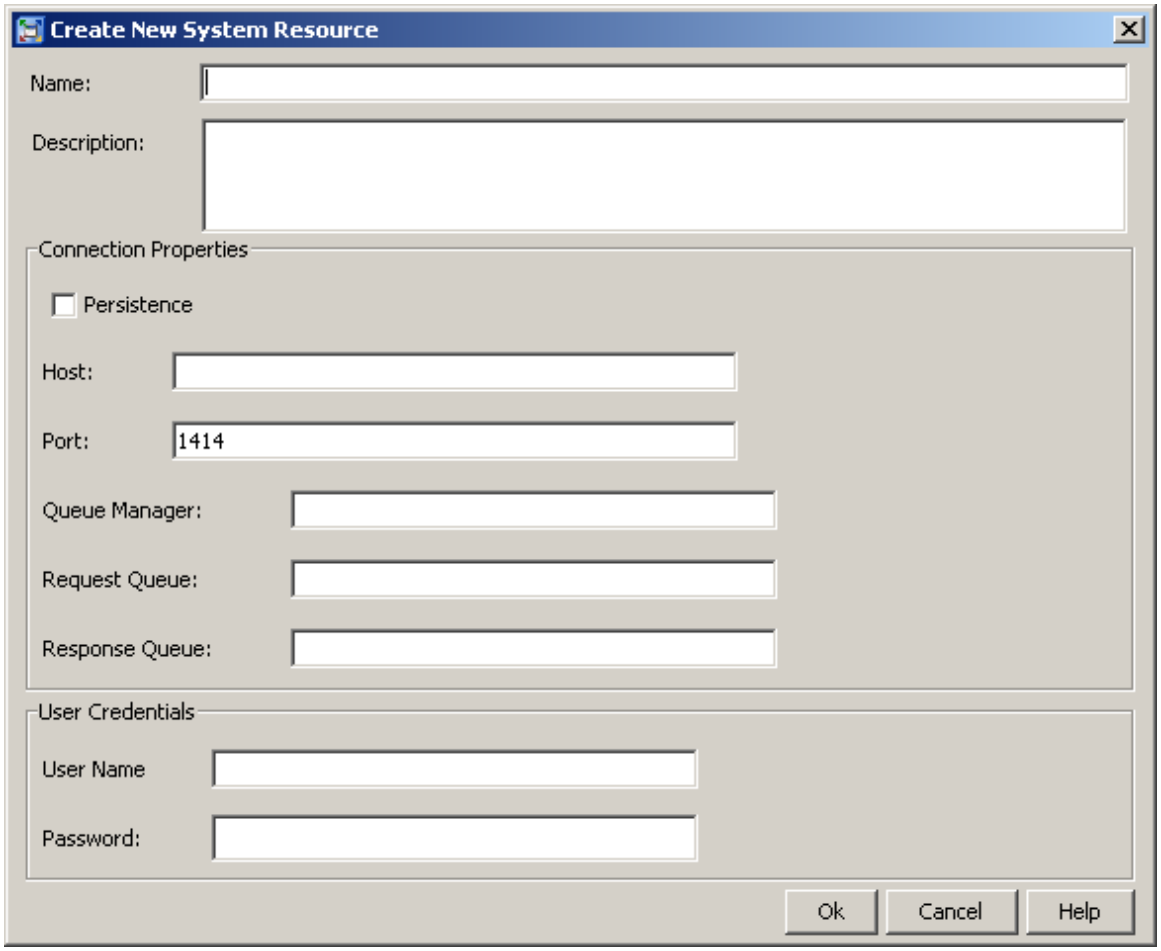

3. Select **New System Resource** from the drop-down menu.



4. Select SASConnection.




5. Complete any required fields in the dialog box that appears.




The terms and definitions that follow are also listed in the Help for this dialog box.

**Name**

specifies the name of the system resource. Host has a 60–character maximum length. Spaces are allowed.

**Description**

(optional) might include the SAS activity or server cluster for which you plan to use this SAS connection. Description has a 200–character maximum length.

**Persistence**

a check mark indicates that the messages are not lost if the queue connection is broken. The messages remain in existence when the queue connection is restored.

**Note:** If the box is checked, then performance is degraded.

**Host**

specifies the URI or IP address of the WebSphere MQ server.

**Port**

specifies the port on which the WebSphere MQ server is listening.

**Queue Manager**

specifies the name of the queue manager that is used by the WebSphereMQ server. Your SAS installation consultant typically uses a script called SetupMQ (for WebSphereMQ) to create a queue manager called QM_RTDM.

**Request Queue**

specifies the name of the queue that is used to send messages to the server or server cluster that performs the SAS activity that is assigned to this system resource.

**Response Queue**

specifies the name of the queue that is used to receive messages from the server or server cluster that performs the SAS activity that is assigned to this system resource.

**User Name**

(optional) specifies your user name if security is enabled on the request and response queues.

**Password**

(optional) specifies your password if security is enabled on the request and response queues.

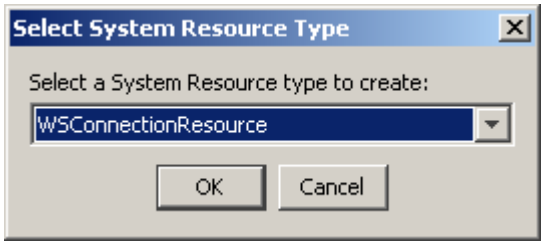
After you click **OK**, the new SAS Connection system resource should appear in the repository.

## Specifying a New System Resource as a Web Service Connection

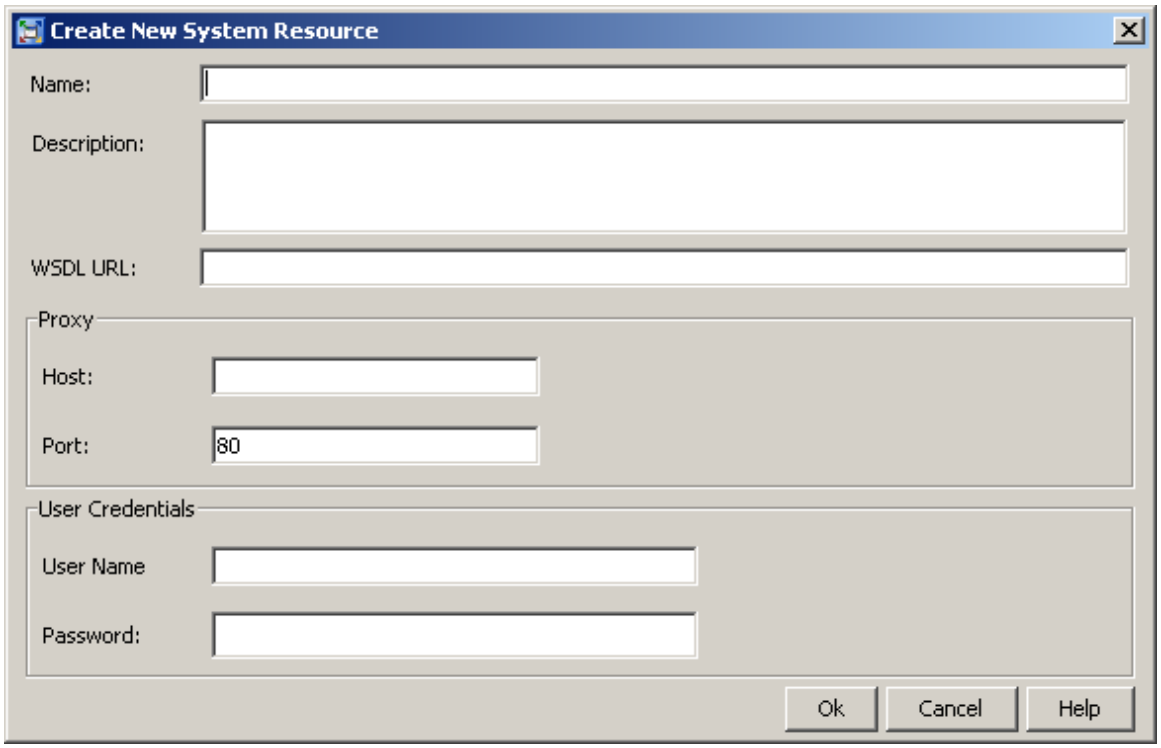
The Web Service Activity type does not use the SAS server tier for processing. Instead, it makes a direct request to the Web service as specified by the Web Service Connection system resource.

To specify a Web service connection as a system resource, follow steps in the previous topic (Specifying a New System Resource as a SAS Connection), and continue with these steps:

1. Select WSConnectionResource.




2. Complete any required fields in the dialog box that appears.




The terms and definitions that follow are also listed in the Help for this dialog box.

**Name**
specifies the name of the system resource. Name has a 60-character maximum length; spaces are allowed.

**Description**
(optional) might specify the Web Service activity that you plan to use this system resource for. Description has a 200-character maximum length.

**WSDL URL**
: (required) specifies the URL of the target Web service. If the WSDL URL begins with **https**, then the User Name and Password fields are also required.

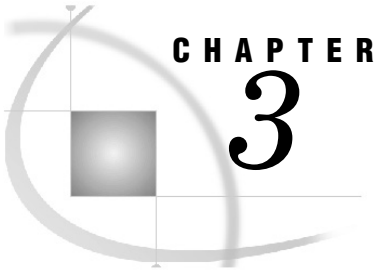
**Host**
: (optional) specifies the proxy server that forwards client requests to other servers. See your system administrator for whether your installation uses a proxy server, and if so, what host name you should use.

**Port**
: (optional) specifies the port that is used by the proxy server.

**User Name**
: If the WSDL URL begins with **https** (indicating that security is enabled), then this field specifies your user name.

**Password**
: If the WSDL URL begins with **https** (indicating that security is enabled), then this field specifies your user password.

After you click **OK**, the new Web Service Connection system resource should appear in the repository.



CHAPTER 3

# SAS Real-Time Decision Manager Activities



## Overview

SAS Real-Time Decision Manager provides a rich set of activities that are suitable for constructing decision flows that automate real-time decisions and actions. Activities perform work actions, such as executing SAS programs on a SAS Server, storing and accessing information from a relational database, sending Web service requests to external systems, executing Scoring Models, etc.

In the rare event that your organization has a special processing need not covered by the provided activity set, new activities may be added. This is accomplished by developing custom SAS code and publishing it into the SAS Real-Time Decision Manager environment. The activity publishing step assembles the metadata necessary for the activity to be recognized by a SAS Real-Time Decision Manager engine and to be rendered in SAS Customer Intelligence Studio.

SAS Real-Time Decision Manager contains five basic classifications of configurable activities:

- SAS Activity
- Web Service Activity
- General I/O Activity

- Scoring Activity
- Code Activity

The activity type that can be used to extend SAS Real-Time Decision Manager functionality is known as the SAS Activity. A SAS Activity consists of a SAS program and an Activity XML file that describes the methods supported by that Activity and the System Resources used by that Activity.

SAS DATA step and macro language programming skills are required to develop SAS code that runs as an activity. For assistance with custom activity development or publishing, contact your on-site SAS support personnel.

# SAS Activities

## Efficiency Considerations

Efficiency is a primary consideration when developing SAS activity code. If your decision flows are required to provide an immediate response, avoid implementing long-running processes. Avoid joins, non-indexed searches, and other expensive database queries. Keep in mind that a decision flow executes no faster than the cumulative speeds of the activities it contains.

Also, avoid using macro code in any SAS activity code, because using macro code could add complexity to debugging custom activity code.

Two versions of custom SAS activity code are supported: macro and DATA step. They are described in the following sections.

## Single Datastep Server (SDS)

The advantage of the single DATA step version is its increased performance. This version uses a DATA step fragment as the custom activity code. All of the code is compiled once in a single DATA step and increased performance is achieved by avoiding a DATA step boundary.

### The DATA Step Code

Custom activity code that is run in this version must be in this form:

```
sas_program_name:
      [SAS Program Data Step Fragment]
      return;
```

In the first line of the code, *sas_program_name* must be replaced with the name of your activity program. Replace *[SAS Program DATA Step Fragment]* with your custom program code. This code must be executable within a DATA step.

Input values are passed into the program through a SAS hash object called sc_input_values. This hash object uses a string key and has two data variables:

- sc_string contains values of type string
- sc_number contains numeric values

The object is put in the appropriate data variable based on the type of the input value.

## *Input and Output Variables*

Input and Output variables are passed through SAS hash objects. The SDS generic code handles decoding the input values into the input hash object (sc_input_values), and encoding output values based on the output hash object (sc_output_values).

**Input**

The sc_input_values hash object consists of three key values and a single data variable. The declaration can be found in the sc_variables.sas file in the SAS Real-Time Decision Manager installation. Its form is as follows:

```
declare hash sc_input_values();
sc_rc = sc_input_values.defineKey('sc_name', 'sc_row_number',
'sc_column_name');
sc_rc = sc_input_values.defineData('sc_data_value');
sc_rc = sc_input_values.defineDone();
```

The following are descriptions of the variables:

- sc_name – This is a character value that contains the name of the input variable.
- sc_row_number – This numeric value is only valid for table and array variable types. For scalar values, this key is always set to missing (.). This key indicates which row of the table or array will be retrieved when a find() call is made.
- sc_column_name – This is a character value that is only valid for the table variable type. For scalar and array values, this key is always missing (.). This key indicates which column of the table will be retrieved when a find() call is made.
- sc_data_value – This is a character value that contains the actual variable data. The data is always stored as a character. To convert the character to a numeric, assign the value to a DATA step variable that is defined as a numeric. SAS will automatically convert it (See examples).

## *Example: How to Use a SAS Real-Time Decision Manager Scalar Variable*

Scalar input values include Int, Float, String, Boolean, and DateTime.

*Int*

A numeric value in SAS. Assigning the sc_data_value to a DATA step variable that is declared as a numeric will cause SAS to automatically convert the value.

```
length myInt 8; /* Declare myInt as a numeric */
sc_name='myInt';
sc_row_number=.;
sc_column_name='';
```

```
sc_input_values.find();
myInt=sc_data_value; /* Automatic conversion */
```

or:

```
length myInt 8; /* Declare myInt as a numeric */
sc_input_values.find(key: 'myInt', key: ., key: '');
myInt=sc_data_value; /* Automatic conversion */
```

*Float*

A numeric value in SAS. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
length myFloat 8; /* Declare myFloat as a numeric */
sc_name='myFloat';
sc_row_number=.;
sc_column_name='';
sc_input_values.find();
myFloat=sc_data_value; /* Automatic conversion */
```

or:

```
length myFloat 8; /* Declare myFloat as a numeric */
sc_input_values.find(key: 'myFloat', key: ., key: '');
myFloat=sc_data_value; /* Automatic conversion */
```

*String*

A character value in SAS. Be sure that the character it is assigned to has a length that is as large as the largest expected string. If the potential length of a string is unknown, the largest character variable length may be used, it is 32767.

**Note:** Excessive use of character variables of this maximum size will have performance and memory usage impacts. If this size character value is needed, it is best to declare a single variable of that length and reuse it.

```
length myString $ 32767; /* This is the largest valid character
variable size */
sc_name='myString';
sc_row_number=.;
sc_column_name='';
sc_input_values.find();
myString=sc_data_value;
```

or:

```
 length myString $ 32767; /* This is the largest valid character
variable size */
 sc_input_values.find(key: 'myString', key: ., key: '');
 myString=sc_data_value;
```

*Boolean*

A numeric value in SAS. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. A value of 1 indicates true and a value of 0 indicates false.

```
 length myBoolean 8; /* Declare myBoolean as a numeric */
 sc_name='myBoolean';
 sc_row_number=.;
 sc_column_name='';
 sc_input_values.find();
 myBoolean=sc_data_value; /* Automatic conversion */
```

or:

```
 length myBoolean 8; /* Declare myBoolean as a numeric */
 sc_input_values.find(key: 'myBoolean', key: ., key: '');
 myBoolean=sc_data_value; /* Automatic conversion */
```

*DateTime*

A numeric value in SAS. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. The numeric will represent a SAS DateTime value.

```
 length myDateTime 8; /* Declare myDateTime as a numeric */
 sc_name='myDateTime';
 sc_row_number=.;
 sc_column_name='';
 sc_input_values.find();
 myDateTime=sc_data_value; /* Automatic conversion */
```

or:

```
 length myDateTime 8; /* Declare myDateTime as a numeric */
 sc_input_values.find(key: 'myDateTime', key: ., key: '');
 myDateTime=sc_data_value; /* Automatic conversion */
```

### *Example: How to Use a SAS Real-Time Decision Manager Array Variable*

Real-Time Decision Manager arrays are more complex objects then scalars. They are added to the input hash object using the name of the array variable and the row (index) into that array for the desired value as keys. The number of rows in the array can be found using an sc_row_number of missing and a column_name of num.rows. The array type can be retrieved by using an sc_row_number of missing and a column_name of array.type. In most cases, the activity writer will know the type of array ahead of time in order to use the values correctly. The information is provided in the hash object for the rare case when it is needed.

The following is an example of how to retrieve the number of rows in the array:

```
length numRows 8; /* Declare numRows as a numeric */
 sc_name='myArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */
```

or:

```
 length numRows 8; /* Declare numRows as a numeric */
 sc_input_values.find(key: 'myArray', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
```

The following is an example of how to retrieve the array type (a value of Int, Float, String, Boolean, or DateTime):

```
 length arrayType $ 8; /* The largest type name is 8 characters */
 sc_name='myArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_input_values.find();
 arrayType=sc_data_value;
```

or:

```
 length arrayType $ 8; /* The largest type name is 8 characters */
 sc_input_values.find(key: 'myArray', key: ., key: 'array.type');
 arrayType=sc_data_value;
```

Integer Array

This uses numerics for the array values. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
 length myInt numRows 8; /* Declare myInt and numrows as numerics */
 /* Retrieve the number of rows */
 sc_name='myIntArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */
 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find();
     myInt=sc_data_value; /* Automatic conversion */
 end;
```

or:

```
 length myInt numRows 8; /* Declare myInt and numrows as numerics */
 sc_input_values.find(key: 'myIntArray', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myIntArray', key: sc_row_number,
key: '');
     myInt=sc_data_value; /* Automatic conversion */
 end;
```

Float Array

This uses numerics for the array values. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
 length myFloat numRows 8; /* Declare myFloatand numRows as numerics
*/
 /* Retrieve the number of rows */
 sc_name='myFloatArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */
 sc_column_name=''; /* Clear the column name */
```

```
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find();
     myFloat=sc_data_value; /* Automatic conversion */
 end;
```

or:

```
 length myFloat numRows 8; /* Declare myFloat and numRows as
numerics */
 sc_input_values.find(key: 'myIntArray', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myIntArray', key: sc_row_number,
key: '');
     myInt=sc_data_value; /* Automatic conversion */
 end;
```

String Array

This uses character values in SAS. Assigning the sc_data_value to a DATA step variable declared as a character will cause SAS to automatically convert the value. Be sure the character variables length is as large as the largest expected string. If the potential length of a string is unknown, the largest character variable length can be used. It is 32767.

**Note:** Excessive use of character variables of this maximum size will have performance and memory usage impacts. If this size character value is needed, it is best to declare a single variable of that length and reuse it.

```
 length numRows 8; /* Declare numRows as numeric */
 length myString $ 32767; /* This is the largest valid character
variable size */
 /* Retrieve the number of rows */
 sc_name='myStringArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */
 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
```

```
 do sc_row_number=1 to numRows;
     sc_input_values.find();
     myStrimg=sc_data_value;
 end;
```

or:

```
 length numRows 8; /* Declare numRows as numeric */
 length myString $ 32767; /* This is the largest valid character
variable size */
 sc_input_values.find(key: 'myStringArray', key: ., key:
'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myStringArray', key: sc_row_number,
key: '');
     myString=sc_data_value; /* Automatic conversion */
 end;
```

Boolean Array

This uses numeric values in SAS. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. A value of 1 indicates true and a value of 0 indicates false.

```
 length myBoolean numRows 8; /* Declare myBoolean and numRows as
numerics */
 /* Retrieve the number of rows */
 sc_name='myBooleanArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */
 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find();
     myBoolean=sc_data_value; /* Automatic conversion */
 end;
```

or:

```
 length myBoolean numRows 8; /* Declare myBoolean and numRows as
numerics */
 sc_input_values.find(key: 'myBooleanArray', key: ., key:
'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myBooleanArray', key: sc_row_number,
key: '');
     myBoolean=sc_data_value; /* Automatic conversion */
 end;
```

DateTime Array

This uses numeric values in SAS. Assigning the sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. The numeric will represent a SAS DateTime value.

```
 length myDateTime numRows 8; /* Declare myDateTime and numRows as
numerics */
 /* Retrieve the number of rows */
 sc_name='myDateTimeArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */
 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find();
     myDateTime=sc_data_value; /* Automatic conversion */
 end;
```

or:

```
 length myDateTime numRows 8; /* Declare myDateTime and numRows as
numerics */
 sc_input_values.find(key: 'myDateTimeArray', key: ., key:
'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myTimeArray', key: sc_row_number,
key: '');
     myDateTime=sc_data_value; /* Automatic conversion */
 end;
```

## *Example: How to Use a SAS Real-Time Decision Manager Table Variable*

Real-Time Decision Manager tables are the most complex object type. They are added to the input hash object using the name of the table variable, the row, and the column name for the desired value as keys. The number of rows in the table can be found using an sc_row_number of missing and a column_name of num.rows. The number of columns in the table can be found using an sc_row_number of missing and a column_name of num.columns. The name of a given column can be retrieved by using an sc_row_number equal to the column number and a column_name value of '' (empty string). The column names are provided for convenience.

**Note:** An empty string consists of two single quotation marks with no space in between.

In most cases the activity code writer would need to know the column names ahead of time in order to make proper use of the values they contain. The type of the column can be found by using an sc_row_number of missing and a column_name value that is the actual column name. Again, in most cases the activity writer would need to know ahead of time what the type of each column is in order to make proper use of the values. However, the information is provided in the hash object for the rare case when it is needed.

For these examples, assume a table named myTable with a column of each scalar type, myInt of type Int, myFloat of type Float, myString of type String, myBoolean of type Boolean, and myDateTime of type DateTime.

### *Metadata Retrieval*

The Table metadata consists of the number of rows, the number of columns, the column names, and the column types. In most cases the activity code writer will already know all of the metadata except the number of rows. However, all of the metadata is provided for cases where it is needed.

The following is an example of how to retrieve the number of rows:

```
length numRows 8; /* Declare numRows as a numeric */
sc_name='myTable';
sc_row_number=.;
sc_column_name='num.rows';
sc_input_values.find();
numRows=sc_data_value; /* Automatic conversion */
```

or:

```
 length numRows 8; /* Declare numRows as a numeric */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */
```

The following is an example of how to retrieve the number of columns:

```
 length numColumns 8; /* Declare numColumns as a numeric */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.columns';
 sc_input_values.find();
 numColumns=sc_data_value; /* Automatic conversion */

or:

 length numColumns 8; /* Declare numColumns as a numeric */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.columns');
 numColumns=sc_data_value; /* Automatic conversion */
```

The following is an example of how to retrieve the column names:

```
 /* Retrieve a single column name */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */
 sc_name='myTable';
 sc_row_number=1; /* Get the name fo the first column */
 sc_column_name='';
 sc_input_values.find();
 columnName=sc_data_value;

or:

 /* Retrieve a single column name */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */
 sc_input_values.find(key: 'myTable', key: 1, key: '');
 columnName=sc_data_value;
 /* Retrieve all of the column names */
 length numColumns 8; /* Declare numColumns as a numeric */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */
```

```
 /* Retrieve the number of columns */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.columns';
 sc_input_values.find();
 numColumns=sc_data_value; /* Automatic conversion */

 sc_column_name=''; /* Clear the column_name */

 /* Loop to retrieve each column name */
 do sc_row_number=1 to numColumns;
     sc_input_values.find();
     columnName=sc_data_value;
 end;
```

or:

```
 /* Retrieve all of the column names */
 length numColumns 8; /* Declare numColumns as a numeric */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.columns');
 numColumns=sc_data_value; /* Automatic conversion */
 do sc_row_number=1 to numColumns;
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'');
     columnName=sc_data_value;
 end;
```

The following is an example of how to retrieve the column types (a value of Int, Float, String, Boolean, or DateTime):

```
 /* Retrieve a single column type */
 length columnType $ 8; /* The largest type name is 8 characters */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='myInt'; /* Find the type of the myInt column */
 sc_input_values.find();
 columnType=sc_data_value; /* Will contain a value of Int */
```

or:

```
 /* Retrieve a single column type */
 length columnType $ 8; /* The largest type name is 8 characters */
 sc_input_values.find(key: 'myTable', key: ., key: 'myInt'); /* Find
the type of the myInt column */
 columnType=sc_data_value; /* Will contain a value of Int */
 /* Retrieve all column types */
 length numColumns columnIndex 8; /* Declare numColumns and
columnIndex as numerics */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */
 length columnType $ 8; /* The largest type name is 8 characters */

 /* Retrieve the number of columns */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.columns';
 sc_input_values.find();
 numColumns=sc_data_value; /* Automatic conversion */

 /* Loop to retrieve the column name and its type */
 do columnIndex=1 to numColumns;
     sc_column_name=''; /* Clear the previous column name */
     sc_row_number=columnIndex; /* Set the row number to the current
index */
     sc_input_values.find(); /* Finds the column name at the given
sc_row_number */
     sc_row_number=.; /* Clear the previous value fo the row number
*/
     sc_column_name=sc_data_value; /* Set the current column name */
     sc_input_values.find(); /* Finds the column type for the given
column name */
     columnType=sc_data_value; /* Will contain the column type */
 end;
```

or:

```
 /* Retrieve all column types */
 length numColumns columnIndex 8; /* Declare numColumns and
columnIndex as numerics */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */
 length columnType $ 8; /* The largest type name is 8 characters */

 /* Retrieve the number of columns */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.columns');
 numColumns=sc_data_value; /* Automatic conversion */
```

```
 /* Loop to retrieve the column name and its type */
 do columnIndex=1 to numColumns;
     sc_column_name=''; /* Clear the previous column name */
     sc_row_number=columnIndex; /* Set the row number to the current
index */
     sc_input_values.find(key: 'myTable', key: columnIndex, key:
''); /* Finds the column name at the given columnIndex */
     sc_input_values.find(key: 'myTable', key: ., key:
sc_data_value); /* Finds the column type for the given column name
*/
     columnType=sc_data_value; /* Will contain the column type */
 end;
```

*Data Retrieval*

The following is an example of how to retrieve a single value:

```
 length myInt 8; /* Declare myInt as a numeric */
 sc_name='myTable';
 sc_row_number=1;
 sc_column_name='myInt';
 sc_input_values.find(); /* Retrieves the value of column myInt, row
1 */
 myInt=sc_data_value;  /* Automatic conversion */
or:
 length myInt 8; /* Declare myInt as a numeric */
 sc_input_values.find(key: 'myTable', key: 1, key: 'myInt'); /*
Retrieves the value of column myInt, row 1 */
 myInt=sc_data_value; /* Automatic conversion */
```

The following is an example of how to retrieve all of the values in a single column:

```
 length numRows myInt 8; /* Declare numRows and myInt as numerics */

 /* Retrieve the number of rows */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */

 /* Loop to retrieve the value for the column at each row */
 sc_column_name='myInt';
```

```
 do sc_row_number=1 to numRows;
     sc_input_values.find(); /* Retrieve the value for the current
row */
     myInt=sc_data_value; /* Automatic conversion */
 end;
```

or:

```
 length myInt 8; /* Declare myInt as a numeric */
 sc_input_values.find(key: 'myTable', key: 1, key: 'myInt'); /*
Retrieves the value of column myInt, row 1 */
 myInt=sc_data_value; /* Automatic conversion */
```

The following is an example of how to retrieve all of the values in a single column:

```
 length numRows myInt 8; /* Declare numRows and myInt as numerics */

 /* Retrieve the number of rows */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */

 /* Loop to retrieve the value for the column at each row */
 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'myInt'); /* Retrieve the value for the current row */
     myInt=sc_data_value; /* Automatic conversion */
 end;
```

The following is an example of how to retrieve all of the values in a single row:

```
 length myInt myFloat myBoolean myDatTime 8; /* Declare numeric
scalar types to hold data for each column */
 length myString $ 32767; /* Declare character scalar to hold data
for Strings */
 sc_name='myTable';
 sc_row_number=1;
 sc_column_name='myInt';
 sc_input_values.find(); /* Retrieves the value of column myInt, row
1 */
 myInt=sc_data_value;  /* Automatic conversion */
 sc_column_name='myFloat';
 sc_input_values.find(); /* Retrieves the value of column myFloat,
row 1 */
 myFloat=sc_data_value;  /* Automatic conversion */
```

```
 sc_column_name='myString';
 sc_input_values.find(); /* Retrieves the value of column myString,
row 1 */
 myString=sc_data_value;
 sc_column_name='myBoolean';
 sc_input_values.find(); /* Retrieves the value of column myBoolean,
row 1 */
 myBoolean=sc_data_value;  /* Automatic conversion */
 sc_column_name='myDateTime';
 sc_input_values.find(); /* Retrieves the value of column
myDateTime, row 1 */
 myDateTime=sc_data_value;  /* Automatic conversion */
```

or:

```
 length myInt myFloat myBoolean myDatTime 8; /* Declare numeric
scalar types to hold data for each column */
 length myString $ 32767; /* Declare character scalar to hold data
for Strings */
 sc_input_values.find(key: 'myTable', key: 1, key: 'myInt'); /*
Retrieves the value of column myInt, row 1 */
 myInt=sc_data_value;  /* Automatic conversion */
 sc_input_values.find(key: 'myTable', key: 1, key: 'myFloat'); /*
Retrieves the value of column myFloat, row 1 */
 myFloat=sc_data_value;  /* Automatic conversion */
 sc_input_values.find(key: 'myTable', key: 1, key: 'myString'); /*
Retrieves the value of column myString, row 1 */
 myString=sc_data_value;
 sc_input_values.find(key: 'myTable', key: 1, key: 'myBoolean'); /*
Retrieves the value of column myBoolean, row 1 */
 myBoolean=sc_data_value;  /* Automatic conversion */
 sc_input_values.find(key: 'myTable', key: 1, key: 'myDateTime'); /*
Retrieves the value of column myDateTime, row 1 */
 myDateTime=sc_data_value;  /* Automatic conversion */
```

The following is an example of how to retrieve all of the values by row:

```
 length myInt myFloat myBoolean myDatTime 8; /* Declare numeric
scalar types to hold data for each column */
 length myString $ 32767; /* Declare character scalar to hold data
for Strings */
 length numRows 8; /* Declare numRows as a numeric */

 /* Retrieve the number of rows */
 sc_name='myTable';
 sc_row_number=.;
```

```
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */

 do sc_row_number=1 to numRows;
     sc_column_name='myInt';
     sc_input_values.find(); /* Retrieves the value of column myInt
for the current row */
     myInt=sc_data_value;  /* Automatic conversion */
     sc_column_name='myFloat';
     sc_input_values.find(); /* Retrieves the value of column
myFloat for the current row */
     myFloat=sc_data_value;  /* Automatic conversion */
     sc_column_name='myString';
     sc_input_values.find(); /* Retrieves the value of column
myString for the current row */
     myString=sc_data_value;
     sc_column_name='myBoolean';
     sc_input_values.find(); /* Retrieves the value of column
myBoolean for the current row */
     myBoolean=sc_data_value;  /* Automatic conversion */
     sc_column_name='myDateTime';
     sc_input_values.find(); /* Retrieves the value of column
myDateTime for the current row */
     myDateTime=sc_data_value;  /* Automatic conversion */
 end;
```

or:

```
 length myInt myFloat myBoolean myDatTime 8; /* Declare numeric
scalar types to hold data for each column */
 length myString $ 32767; /* Declare character scalar to hold data
for Strings */
 length numRows 8; /* Declare numRows as a numeric */

 /* Retrieve the number of rows */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */

 do sc_row_number=1 to numRows;
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'myInt'); /* Retrieves the value of column myInt for the current row
*/
     myInt=sc_data_value;  /* Automatic conversion */
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'myFloat'); /* Retrieves the value of column myFloat for the current
row */
```

```
     myFloat=sc_data_value;  /* Automatic conversion */
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'myString'); /* Retrieves the value of column myString for the
current row */
     myString=sc_data_value;
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'myBoolean'); /* Retrieves the value of column myBoolean for the
current row */
     myBoolean=sc_data_value;  /* Automatic conversion */
     sc_input_values.find(key: 'myTable', key: sc_row_number, key:
'myDateTime'); /* Retrieves the value of column myDateTime for the
current row */
     myDateTime=sc_data_value;  /* Automatic conversion */
 end;
```

The following is an example of how to retrieve all values by row using column metadata:

```
 length myNumeric 8; /* Declare myNumeric to hold data for numeric
columns */
 length myString $ 32767; /* Declare character scalar to hold data
for Strings */
 length numRows rowIndex numColumns columnIndex 8; /* Declare
numRows, rowIndex, numColumns, and columnIndex as numerics */

 /* Retrieve the number of rows */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_input_values.find();
 numRows=sc_data_value; /* Automatic conversion */

 /* Retrieve the number of columns */
 sc_column_name='num.columns';
 sc_input_values.find();
 numColumns=sc_data_value; /* Automatic conversion */

 /* Read the table one row at a time */
 do rowIndex=1 to numRows;
     /* Retrieve each column name and type */
     do columnIndex=1 to numColumns;
         sc_column_name=''; /* Clear the previous column name */
         sc_row_number=columnIndex; /* Set the row number to the
current column index */
         sc_input_values.find(); /* Finds the column name at the
given sc_row_number */
         sc_row_number=.; /* Clear the previous value of the row
number */
```

```
         sc_column_name=sc_data_value; /* Set the current column
name */
         sc_input_values.find(); /* Finds the column type for the
given column name */
         columnType=sc_data_value; /* Will contain the column type
*/
         sc_row_number=rowIndex; /* Sets the row number to the
current row index */
         sc_input_values.find(); /* Finds the value for the current
row, column */

         /* Put the value into the appropriate variable type */
         if (columnType EQ 'String') do;
             myString=sc_data_value;
         end;
         else do;
             myNumeric=sc_data_value; /* Automatic conversion */
         end;
     end;
 end;
```

or:

```
 length myNumeric 8; /* Declare myNumeric to hold data for numeric
columns */
 length myString $ 32767; /* Declare character scalar to hold data
for Strings */
 length numRows rowIndex numColumns columnIndex 8; /* Declare
numRows, rowIndex, numColumns, and columnIndex as numerics */
 length columnName $ 255; /* Declare columnName with the maximum
length for a name, 255 */

 /* Retrieve the number of rows */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.rows');
 numRows=sc_data_value; /* Automatic conversion */

 /* Retrieve the number of columns */
 sc_input_values.find(key: 'myTable', key: ., key: 'num.columns');
 numColumns=sc_data_value; /* Automatic conversion */

 /* Read the table one row at a time */
 do rowIndex=1 to numRows;
     /* Retrieve each column name and type */
     do columnIndex=1 to numColumns;
         sc_input_values.find(key: 'myTable', key: columnIndex, key:
''); /* Finds the column name at the given sc_row_number */
```

```
        columnName=sc_data_value; /* Will contain the column name
*/
        sc_input_values.find(key: 'myTable', key: ., key:
columnName); /* Finds the column type for the given column name */
        columnType=sc_data_value; /* Will contain the column type
*/
        sc_input_values.find(key: 'myTable', key: rowIndex, key:
columnName); /* Finds the value for the current row, column */

        /* Put the value into the appropriate variable type */
        if (columnType EQ 'String') do;
            myString=sc_data_value;
        end;
        else do;
            myNumeric=sc_data_value; /* Automatic conversion */
        end;
    end;
end;
```

**Output**

The sc_output_values hash object consists of three key values and a single data variable. This matches the form of the input hash object. Two separate hash objects are used, so that if an output variable uses the same name as an input variable it will not be accidentally overwritten. The declaration can be found in the sc_variables.sas file in the Real-Time Decision Manager installation. Its form is as follows:

```
declare hash sc_output_values();
 sc_rc = sc_input_values.defineKey('sc_name', 'sc_row_number',
'sc_column_name');
 sc_rc = sc_input_values.defineData('sc_data_value');
 sc_rc = sc_input_values.defineDone();
```

- sc_name – This is a character value that contains the name of the output variable.
- sc_row_number – This is numeric value that is only valid for table and array variable types. For scalar values, this key is always set to missing (.). This key indicates which row of the table or array will be set when an add() or replace() call is made.
- sc_column_name – This is a character value that is only valid for the table variable type. For scalar and array values, this key is always missing (.). This key indicates which column of the table will be set when an add() or replace() call is made.
- sc_data_value – This is a character value that contains the actual variable data. The data is always stored as a character. SAS will automatically convert numeric values to character when the assignment to sc_data_value is made (see examples).

## *Example: How to Use a SAS Real-Time Decision Manager Scalar Variable*

Scalar output values include Int, Float, String, Boolean, and DateTime.

*Int*

A numeric value in SAS. Assigning sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
 length myInt 8; /* Declare myInt as a numeric */
 sc_name='myInt';
 sc_row_number=.;
 sc_column_name='';
 myInt=8;
 sc_data_value=myInt; /* Automatic conversion */
 sc_output_values.replace();
```

or:

```
 length myInt 8; /* Declare myInt as a numeric */
 myInt=8;
 sc_data_value=myInt; /* Automatic conversion */
 sc_output_values.replace(key: 'myInt', key: ., key: '', data:
sc_data_value);
```

or:

```
 sc_output_values.replace(key: 'myInt', key: ., key: '', data: '8');
```

*Float*

A numeric value in SAS. Assigning sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
 length myFloat 8; /* Declare myFloat as a numeric */
 sc_name='myFloat';
 sc_row_number=.;
 sc_column_name='';
 myFloat=5.05;
 sc_data_value=myFloat; /* Automatic conversion */
 sc_output_values.replace();
```

or:

```
 length myFloat 8; /* Declare myFloat as a numeric */
 myFloat=5.05;
 sc_data_value=myFloat; /* Automatic conversion */
 sc_output_values.replace(key: 'myFloat', key: ., key: '', data:
sc_data_value);
```

or:

```
 sc_output_values.replace(key: 'myFloat', key: ., key: '', data:
'5.05');
```

*String*

A character value in SAS.

```
 sc_name='myString';
 sc_row_number=.;
 sc_column_name='';
 sc_data_values='This is a string.';
 sc_output_values.replace();
```

or:

```
 sc_output_values.replace(key: 'myString', key: ., key: '', data:
'This is a string.');
```

*Boolean*

A numeric value in SAS. Assigning sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. A value of 1 indicates true and a value of 0 indicates false.

```
 length myBoolean 8; /* Declare myBoolean as a numeric */
 sc_name='myBoolean';
 sc_row_number=.;
 sc_column_name='';
 myBoolean=1;
 sc_data_value=myBoolean; /* Automatic conversion */
 sc_output_values.replace();
```

or:

```
 length myBoolean 8; /* Declare myBoolean as a numeric */
 myBoolean=1;
 sc_data_value=myBoolean; /* Automatic conversion */
 sc_output_values.replace(key: 'myBoolean', key: ., key: '', data:
sc_data_value);
```

or:

```
 sc_output_values.replace(key: 'myBoolean', key: ., key: '', data:
'1');
```

*DateTime*

A numeric value in SAS. Assigning sc_data_value to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. The numeric will represent a SAS DateTime value.

```
 length myDateTime 8; /* Declare myDateTime as a numeric */
 sc_name='myDateTime';
 sc_row_number=.;
 sc_column_name='';
 myDateTime='10OCT2006:04:03:16'DT;
 sc_data_value=myDatTime; /* Automatic conversion */
 sc_output_values.replace();
```

or:

```
 length myDateTime 8; /* Declare myDateTime as a numeric */
 myDateTime='10OCT2006:04:03:16'DT;
 sc_data_value=myDateTime; /* Automatic conversion */
 sc_output_values.replace(key: 'myDateTime', key: ., key: '', data:
sc_data_value);
```

## *Example: How to Use a SAS Real-Time Decision Manager Array Variable*

Real-Time Decision Manager arrays are more complex objects then scalars. They are added to the output hash object using the name of the array variable and the row (index) into that array for the desired value as keys. The number of rows in the array must be set using an sc_row_number of missing and a column_name of num.rows. The array type must be set using an sc_row_number of missing and a column_name of array.type.

**Note:** The array type is not case sensitive. However, it must have a valid value of Int, Float, String, Boolean, or DateTime.

The following is an example of how to set the number of rows in the array:

```
 length numRows 8; /* Declare numRows as a numeric */
 numRows=4;
 sc_name='myArray';
 sc_row_number=.;
 sc_column_name='num.rows';
```

```
 sc_data_value=numRows; /* Automatic conversion */
 sc_output_values.add();

or:

 length numRows 8; /* Declare numRows as a numeric */
 numRows=4;
 sc_data_value=numRows; /* Automatic conversion */
 sc_output_values.add(key: 'myArray', key: ., key: 'num.rows', data:
sc_data_value);
```

The following is an example of how to set the array type (a value of Int, Float, String, Boolean, or DateTime):

```
 length arrayType $ 8; /* The largest type name is 8 characters */
 arrayType='Int'
 sc_name='myArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_output_values.add();
or:
 sc_output_values.add(key: 'myArray', key: ., key: 'array.type',
data: 'Int');
```

### Integer Array

Real-Time Decision Manager array Int uses numerics for the array values. The numeric value must be assigned to a character variable before being stored in the hash object.

```
 /* Set the number of rows */
 sc_name='myIntArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add();

 /* Set the array type */
 sc_name='myIntArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_data_value='Int';
 sc_output_values.add();

 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
```

```
 do sc_row_number=1 to 4;
     sc_data_value=sc_row_number; /* Automatic conversion */
     sc_output_values.add();
 end;
```

or:

```
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add(key: 'myIntArray', key: ., key: 'num.rows',
data: sc_data_value);
 sc_output_values.add(key: 'myIntArray', key: ., key: 'array.type',
data: 'Int');

 /* Iterate over all of the values in this array */
 do sc_row_number=1 to 4;
     sc_data_value=sc_row_number; /* Automatic conversion */
     sc_output_values.add(key: 'myIntArray', key: sc_row_number,
key: '', data: sc_data_value);
 end;
```

### Float Array

Real-Time Decision Manager array float uses numerics for the array values. The numeric value must be assigned to a character variable before being stored in the hash object.

```
 /* Set the number of rows */
 sc_name='myFloatArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add();

 /* Set the array type */
 sc_name='myFloatArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_data_value='Float';
 sc_output_values.add();

 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to 4;
     sc_data_value=15.6 + sc_row_number; /* Automatic conversion */
     sc_output_values.add();
 end;
```

or:

```
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add(key: 'myFloatArray', key: ., key: 'num.rows',
data: sc_data_value);
 sc_output_values.add(key: 'myFloatArray', key: ., key:
'array.type', data: 'Float');

 /* Iterate over all of the values in this array */
 do sc_row_number=1 to numRows;
     sc_data_value=15.6 + sc_row_number; /* Automatic conversion */
     sc_input_values.find(key: 'myFloatArray', key: sc_row_number,
key: '', data: sc_data_value);
 end;
```

### String Array

Real-Time Decision Manager array string uses character values in SAS.

```
 /* Set the number of rows */
 sc_name='myStringArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add();

 /* Set the array type */
 sc_name='myStringArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_data_value='String';
 sc_output_values.add();

 sc_column_name=''; /* Clear the column name */
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to 4;
     sc_data_value='String' || sc_row_number;
     sc_output_values.add();
 end;
```

or:

```
 sc_data_value=4; /* Automatic conversion */
```

```
 sc_output_values.add(key: 'myStringArray', key: ., key: 'num.rows',
data: sc_data_value);
 sc_output_values.add(key: 'myStringArray', key: ., key:
'array.type', data: 'String');

 /* Iterate over all of the values in this array */
 do sc_row_number=1 to 4;
     sc_data_value='String' || sc_row_number;
     sc_output_values.add(key: 'myStringArray', key: sc_row_number,
key: '', data: sc_data_value);
 end;
```

### Boolean Array

Real-Time Decision Manager array boolean uses numeric values in SAS. The numeric value must be assigned to a character variable before being stored in the hash object. A value of 1 indicates true and a value of 0 indicates false.

```
 /* Set the number of rows */
 sc_name='myBooleanArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_data_value=2; /* Automatic conversion */
 sc_output_values.add();

 /* Set the array type */
 sc_name='myBooleanArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_data_value='Boolean';
 sc_output_values.add();

 sc_column_name=''; /* Clear the column name */
 sc_row_numer=1;
 sc_data_value=1; /* True */
 sc_output_values.add();
 sc_row_numer=2;
 sc_data_value=0; /* False */
 sc_output_values.add();

or:

 sc_data_value=2; /* Automatic conversion */
 sc_output_values.add(key: 'myBooleanArray', key: ., key:
'num.rows', data: sc_data_value);
```

```
 sc_output_values.add(key: 'myBooleanArray', key: ., key:
'array.type', data: 'Boolean');
 sc_output_values.add(key: 'myBooleanArray', key: 1, key: '', data:
'1'); /* True */
 sc_output_values.add(key: 'myBooleanArray', key: 2, key: '', data:
'0'); /* False */
```

### DateTime Array

Real-Time Decision Manager array DateTime uses numeric values in SAS. The numeric value must be assigned to a character variable before being stored in the hash object. The numeric must represent a SAS DateTime value.

```
 length myDateTime 8; /* Declare myDateTime as numeric */
 /* Set the number of rows */
 sc_name='myDateTimeArray';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add();

 /* Set the array type */
 sc_name='myDateTimeArray';
 sc_row_number=.;
 sc_column_name='array.type';
 sc_data_value='DateTime';
 sc_output_values.add();

 sc_column_name=''; /* Clear the column name */
 myDateTime='10OCT2006:04:03:16'DT;
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to 4;
     sc_data_value=myDateTime + ((24 * sc_row_number) * 60 * 60); /*
Add a day times the row number, automatic conversion */
     sc_output_values.add();
 end;

or:

 length myDateTime 8; /* Declare myDateTime as numeric */
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add(key: 'myDateTimeArray', key: ., key:
'num.rows', data: sc_data_value);
 sc_output_values.add(key: 'myDateTimeArray', key: ., key:
'array.type', data: 'DateTime');

 myDateTime='10OCT2006:04:03:16'DT;
```

```
 /* Iterate over all of the values in this array */
 do sc_row_number=1 to 4;
     sc_data_value=myDateTime + ((24 * sc_row_number) * 60 * 60); /*
Add a day times the row number, automatic conversion */
     sc_output_values.add(key: 'myTimeArray', key: sc_row_number,
key: '', data: sc_data_value);
 end;
```

## *Example: How to Use a SAS Real-Time Decision Manager Table Variable*

Real-Time Decision Manager tables are the most complex object type. Table values are added to the output hash object using the name of the table variable, the row, and the column name as keys. The number of rows in the table must be set using an sc_row_number of missing and a column_name of num.rows. The number of columns in the table must be set using an sc_row_number of missing and a column_name of num.columns. The name of a given column must be set using an sc_row_number equal to the column number and a column_name value of '' (empty string).

**Note:** An empty string consists of two single quotation marks with no space in between.

The type of the column must be set using an sc_row_number of missing and a column_name value that is the actual column name. The table metadata is needed by the Real-Time Decision Manager SAS connection code in order to properly return the table to the middle tier. If it is not set correctly the table will not transfer correctly.

For these examples assume a table named myTable with a column of each scalar type, myInt of type Int, myFloat of type Float, myString of type String, myBoolean of type Boolean, and myDateTime of type DateTime.

### *Setting Metadata*

The table metadata consists of the number of rows, the number of columns, the column names, and the column types. These values must be set in order for the table to properly be turned to the middle tier.

The following is an example of how to set the number of rows:

```
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.rows';
 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add();

or:

 sc_data_value=4; /* Automatic conversion */
 sc_output_values.add(key: 'myTable', key: ., key: 'num.rows', data:
sc_data_value);
```

The following is an example of how to set the number of columns:

```
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.columns';
 sc_data_value=5; /* Automatic conversion */
 sc_output_values.add();
```

or:

```
 sc_data_value=5; /* Automatic conversion */
 sc_output_values.add(key: 'myTable', key: ., key: 'num.columns',
data: sc_data_value);
```

The following is an example of how to set the column names:

```
 /* Set a single column name */
 sc_name='myTable';
 sc_row_number=1; /* Set the name fo the first column */
 sc_column_name='';
 sc_data_values='myInt';
 sc_output_values.add();
```

or:

```
 /* Set a single column name */
 sc_output_values.add(key: 'myTable', key: 1, key: '', Data:
'myInt');
 /* Set all of the column names */

 /* Set the number of columns */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='num.columns';
 sc_data_value=5; /* Automatic conversion */
 sc_output_values.add();

 sc_column_name=''; /* Clear the column_name */

 /* Set each column name */
 sc_row_number=1;
 sc_data_value='myInt';
 sc_output_values.add();
 sc_row_number=2;
 sc_data_value='myFloat';
 sc_output_values.add();
```

```
 sc_row_number=3;
 sc_data_value='myString';
 sc_output_values.add();
 sc_row_number=4;
 sc_data_value='myBoolean';
 sc_output_values.add();
 sc_row_number=5;
 sc_data_value='myDateTime';
 sc_output_values.add();
```

or:

```
 /* Set all of the column names */
 sc_data_value=5; /* Automatic conversion */
 sc_output_values.add(key: 'myTable', key: ., key: 'num.columns',
key: sc_data_value);
 sc_output_values.add(key: 'myTable', key: 1, key: '', 'myInt');
 sc_output_values.add(key: 'myTable', key: 2, key: '', 'myFloat');
 sc_output_values.add(key: 'myTable', key: 3, key: '', 'myString');
 sc_output_values.add(key: 'myTable', key: 4, key: '', 'myBoolean');
 sc_output_values.add(key: 'myTable', key: 5, key: '',
'myDateTime');
```

The following is an example of how to set the column types (a value of Int, Float, String, Boolean, or DateTime):

```
 /* Set a single column type */
 sc_name='myTable';
 sc_row_number=.;
 sc_column_name='myInt'; /* Set the type of the myInt column */
 sc_data_value='Int';
 sc_output_values.add();
```

or:

```
 /* Set a single column type */
 sc_output_values.add(key: 'myTable', key: ., key: 'myInt', data:
'Int'); /* Set the type of the myInt column */
 /* Set all column types */
 sc_row_number=.; /* Clear any previous value of the row number */

 sc_column_name='myInt'; /* Set the column name */
 sc_data_value='Int'; /* Set the column type */
 sc_output_values.add(); /* Sets the column type for myInt */
```

```
 sc_column_name='myFloat'; /* Set the column name */
 sc_data_value='Float'; /* Set the column type */
 sc_output_values.add(); /* Sets the column type for myFloat */

 sc_column_name='myString'; /* Set the column name */
 sc_data_value='String'; /* Set the column type */
 sc_output_values.add(); /* Sets the column type for myString */

 sc_column_name='myBoolean'; /* Set the column name */
 sc_data_value='Boolean'; /* Set the column type */
 sc_output_values.add(); /* Sets the column type for myBoolean */

 sc_column_name='myDateTime'; /* Set the column name */
 sc_data_value='DateTime'; /* Set the column type */
 sc_output_values.add(); /* Sets the column type for myDateTime */
```

or:

```
 /* Set all column types */
 sc_output_values.add(key: 'myTable', key: ., key: 'myInt', data:
'Int'); /* Sets the column type */
 sc_output_values.add(key: 'myTable', key: ., key: 'myFloat', data:
'Float'); /* Sets the column type */
 sc_output_values.add(key: 'myTable', key: ., key: 'myString', data:
'String'); /* Sets the column type */
 sc_output_values.add(key: 'myTable', key: ., key: 'myBoolean',
data: 'Boolean'); /* Sets the column type */
 sc_output_values.add(key: 'myTable', key: ., key: 'myDateTime',
data: 'DateTime'); /* Sets the column type */
```

*Setting Data*

The following is an example of how to set a single value:

```
 sc_name='myTable';
 sc_row_number=1;
 sc_column_name='myInt';
 sc_data_value=16;  /* Automatic conversion */
 sc_output_values.add(); /* Sets the value of column myInt, row 1 */
```

or:

```
 sc_data_value=16;  /* Automatic conversion */
 sc_output_values.add(key: 'myTable', key: 1, key: 'myInt',
data:sc_data_value ); /* Sets the value of column myInt, row 1 */
```

The following is an example of how to set all of the values in a single column:

```
/* Loop to set the value for the column at each row */
sc_column_name='myInt';
do sc_row_number=1 to 4;
    sc_data_value=sc_row_number; /* Automatic conversion */
    sc_output_values.add(); /* Set the value for the current row */
end;
```

or:

```
/* Loop to set the value for the column at each row */
do sc_row_number=1 to 4;
    sc_data_value=sc_row_number; /* Automatic conversion */
    sc_output_values.add(key: 'myTable', key: sc_row_number, key:
'myInt', data: sc_data_value); /* Set the value for the current row
*/
end;
```

The following is an example of how to set all of the values in a single row:

```
sc_name='myTable';
sc_row_number=1;

sc_column_name='myInt';
sc_data_value=16; /* Automatic conversion */
sc_output_values.add(); /* Sets the value of column myInt, row 1 */

sc_column_name='myFloat';
sc_data_value=33.33; /* Automatic conversion */
sc_output_values.add(); /* Sets the value of column myFloat, row 1
*/

sc_column_name='myString';
sc_data_value='Just a String';
sc_output_values.add(); /* Sets the value of column myString, row 1
*/

sc_column_name='myBoolean';
sc_data_value=1; /* Automatic conversion */
sc_output_values.add(); /* Sets the value of column myBoolean, row
1 */

sc_column_name='myDateTime';
sc_data_value='10OCT2006:04:03:16'DT; /* Automatic conversion */
```

```
 sc_output_values.add(); /* Sets the value of column myDateTime, row
1 */
```

or:

```
 sc_output_values.add(key: 'myTable', key: 1, key: 'myInt', data:
'16'); /* Sets the value of column myInt, row 1 */

 sc_output_values.add(key: 'myTable', key: 1, key: 'myFloat', data:
'33.33'); /* Sets the value of column myFloat, row 1 */

 sc_output_values.add(key: 'myTable', key: 1, key: 'myString', data:
'Just a String'); /* Sets the value of column myString, row 1 */

 sc_output_values.add(key: 'myTable', key: 1, key: 'myBoolean',
data: '1'); /* Sets the value of column myBoolean, row 1 */

 sc_data_value='10OCT2006:04:03:16'DT; /* Automatic conversion */
 sc_output_values.add(key: 'myTable', key: 1, key: 'myDateTime',
data: sc_data_value); /* Sets the value of column myDateTime, row 1
*/
```

## *Example: How to Access a SAS Data Set*

It is recommended that any database I/O be done through the use of the general I/O activity. If data is needed for a custom SAS Activity, a general I/O call should be made prior to the custom SAS Activity and then the data should be passed in.

### Installing a SAS Activity DATA Step

The sc_programs.sas file located in *[install dir]*\Applications\RealTimeServerConfig5.3\SASCode must be updated to use the new activity code. This file contains a large if-else block. To add a new program, cut and paste the last else-if section to add a new section to the if statement.

**Note:** Be sure that you are working with the else-if block, not the else-do block.

Change the hard-coded string on the left of the EQ operand to the name of the new method. Also, change the LINK statement within the block to link to the label of the new method.

## *Example*

```
else if lowcase(sc_programName_var) EQ 'foo' then do;
        link foo;
    end;
```

Finally, be sure to add an include statement at the end of the file to include the new activity code.

### *Example*

```
%include sc_files(foo.sas);
```

**Note:** The above example assumes foo.sas is placed in the same directory as the existing SAS Server files. If it is in a different directory, then use the following line:

```
%include '[dir path]/foo.sas';
```

## Testing the SAS Activity DATA Step

The following code is provided to assist in testing new activity code. This test code assumes the new activity code has been placed in a file and labeled as indicated above.

```
data _null_;
      /* include the SC variables for the Hash objects */
      %include '[dir path]/sc_variables.sas';

      /* add input data values to input Hash object */
      sc_name='myString';
sc_data_value='This is my String';
sc_row_number=.;
sc_column_name='';
      sc_input_values.add();
      sc_name='myNumber';
      sc_data_value=19;
      sc_input_values.add();

      /* call the new activity code */
      link foo;

      /* Check the output values */
      sc_output_values.find(key: 'myString', key: ., key: '');
      put 'myString=' sc_data_value;
      sc_output_values.find(key: 'myNumber', key: ., key: '');
      put 'myNumber=' sc_data_value;
goto exit;
%include "[root path]/foo.sas";
exit:
run;
```

## Error Handling (DATA Step)

DATA step variables are used to indicate error conditions in SAS activity code.

- **sc_retCode_var** a numeric error indicator. Any non-zero value indicates an error has occurred.

- **sc_retMsg_var** holds an error string that indicates why the error occurred. This message is written to the SAS Real-Time Decision Manager server log but is not returned to the user.

### Additional Information (DATA Step)

- Be sure to put a return statement at the end of the program when using the single DATA step version.
- There is a single name space in the single DATA step version. Be sure to keep that in mind when creating DATA step variables. It is recommended to use a prefix in variable names to avoid variable name collision.

## Multiple Datastep Server (MDS)

**IMPORTANT:** Limiting the use of MDS activities to the absolute minimum is highly recommended. MDS might not be supported in future releases of Real-Time Decision Manager.

The MDS version of the Real-Time Decision Manager SAS Activity uses a combination of macro code and DATA step code. This is the more flexible of the two versions as it allows for SAS programs containing multiple DATA and PROC steps. The limit of this version is that it is slower because the SAS language processor compiles DATA and PROC steps at each boundary.

### Input and Output Variables

Input and Output variables are passed through SAS Macro variables. These variables are created and deleted for each Activity code run so that previous values will not affect the current execution. Table and Array objects are a special case as their macro variables need to be encoded/decoded in order to be used.

#### *Input*

Input values are supplied as Macro variables.

#### *Example: How to Use a SAS Real-Time Decision Manager Scalar Variable*

Scalar objects include variables of type Int, Float, String, Boolean, and DateTime. A simple symget() call can be used to get the value of the scalar.

*Int*

Real-Time Decision Manager Int is a numeric value in SAS. Assigning the macro variable to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
length myInt 8; /* Declare myInt as a numeric */
myInt=symget('myInt'); /* Automatic conversion */
```

*Float*

Real-Time Decision Manager Float is a numeric value in SAS. Assigning the macro variable to a DATA step variable declared as a numeric will cause SAS to automatically convert the value.

```
length myFloat 8; /* Declare myFloat as a numeric */
myFloat==symget('myFloat'); /* Automatic conversion */
```

*String*

Real-Time Decision Manager string is a character value in SAS. Be sure the character it is assigned to has a length that is as large as the largest expected string. If the potential length of a string is unknown the largest character variable length may be used. It is 32767. Note excessive use of character variables of this maximum size will have performance and memory usage impacts. If this size character value is needed it is best to declare a single variable of that length and reuse it.

```
length myString $ 32767; /* This is the largest valid character
variable size */
myString=symget('myString');
```

*Boolean*

Real-Time Decision Manager Boolean is a numeric value in SAS. Assigning the macro variable to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. A value of 1 indicates true and a value of 0 indicates false.

```
length myBoolean 8; /* Declare myBoolean as a numeric */
myBoolean=symget('myBoolean'); /* Automatic conversion */
```

*DateTime*

RTDM DateTime is a numeric value in SAS. Assigning the macro variable to a DATA step variable declared as a numeric will cause SAS to automatically convert the value. The numeric will represent a SAS DateTime value.

```
length myDateTime 8; /* Declare myDateTime as a numeric */
myDateTime=symget('myDateTime'); /* Automatic conversion */
```

## *Example: How to Use a SAS Real-Time Decision Manager Array Variable*

Real-Time Decision Manager arrays are more complex objects than scalars. The value assigned to the macro variable for an array is an encoded string. Macro code has been provided to decode array values within a DATA step for use. The array is decoded into a hash object that matches the one used for an SDS array. See the input array section of the SDS for more information on how to access an array in the sc_input_variables hash object. To decode the array within a DATA step, the first line in the DATA step must be %sc_encode_decode. This will provide a DATA step fragment that can be accessed by

calling link decode_array. Before calling, decode the value of DATA step variable sc_name must be set to the name of the array to be decoded.

**Decode an array**

```
data _null_;
    /* This must be at the start of the DATA step */
    %sc_encode_decode

    /* The array can be decoded anywhere within the DATA step */
    sc_name='myArray';
    link sc_decode_array;

    /* At this point the array is available in the sc_input_values
hash object */
run;
```

## *Example: How to Use a SAS Real-Time Decision Manager Table Variable*

Real-Time Decision Manager tables are the most complex object type. The value assigned to the macro variable for a table is an encoded string. Macro code has been provided to decode table values within a DATA step for use. The table is decoded into a hash object that matches the one used for an SDS table. See the input table section of the SDS for more information on how to access a table in the sc_input_variables hash object. To decode the table within a data step the first line in the DATA step must be %sc_encode_decode. This will provide a data step fragment that can be accessed by calling link decode_table. Before calling decode the value of DATA step variable sc_name must be set to the name of the table to be decoded.

**Decode a table**

```
data _null_;
    /* This must be at the start of the DATA step */
    %sc_encode_decode

    /* The table can be decoded anywhere within the DATA step */
    sc_name='myTable';
    link sc_decode_table;

    /* At this point the table is available in the sc_input_values
hash object */
run;
```

## *Output*

## *Example: How to Use a SAS Real-Time Decision Manager Scalar Variable*

Scalar objects include variables of type Int, Float, String, Boolean, and DateTime. A simple symputx() call can be used to set the value of the scalar.

*Int*

Real-Time Decision Manager Int is a numeric value in SAS.

```
length myInt 8; /* Declare myInt as a numeric */
myInt=809;
call symputx('myInt', myInt);
```

*Float*

Real-Time Decision Manager float is a numeric value in SAS.

```
length myFloat 8; /* Declare myFloat as a numeric */
myFloat=809;
call symputx('myFloat', myFloat);
```

*String*

Real-Time Decision Manager string is a character value in SAS.

```
length myString $ 32767; /* This is the largest valid character
variable size */
 myString='This is a string.';
 call symputx('myString', myString);
```

*Boolean*

Real-Time Decision Manager Boolean is a numeric value in SAS. A value of 1 indicates true and a value of 0 indicates false.

```
length myBoolean 8; /* Declare myBoolean as a numeric */
myBoolean=1;
call symputx('myBoolean', myBoolean);
```

*DateTime*

Real-Time Decision Manager DateTime is a numeric value in SAS. The numeric will represent a SAS DateTime value.

```
 length myDateTime 8; /* Declare myDateTime as a numeric */
 myDateTime='10OCT2006:04:03:16'DT;
 call symputx('myDateTime', myDateTime);
```

### *Example: How to Use a SAS Real-Time Decision Manager Array Variable*

Real-Time Decision Manager arrays are more complex objects then scalars. Follow the steps in SDS output to fill in sc_output_values with the array metadata and values. Once that is filled in call sc_encode_array to have the array put into a macro variable.

**Encode an array**

```
data _null_;
    /* This must be at the start of the data step */
    %sc_encode_decode

    /* The array can be encoded anywhere within the DATA step */
    sc_name='myArray';
    link sc_encode_array;
run;
```

### *Example: How to Use a SAS Real-Time Decision Manager TableM Variable*

Real-Time Decision Manager tables are the most complex object type. Follow the steps in SDS output to fill in sc_output_values with the table metadata and values. Once that is filled in call sc_encode_table to have the table put into a macro variable.

**Encode a table**

```
data _null_;
    /* This must be at the start of the DATA step */
    %sc_encode_decode

    /* The table can be encoded anywhere within the DATA step */
    sc_name='myTable';
    link sc_encode_table;
run;
```

## Installing the SAS Activity Macro

The name of the file that the program is placed in must be all lowercase. The SAS autocall mechanism is used to load the file, and it uses all lowercase values when searching for the file. The name of the macro itself can contain mixed cases; it is just the filename that must be all lowercase. The file must then be placed manually into the `SASActivityCode` directory.

```
[installDir]\Applications\SASRealTimeServerConfig5.3\SASActivityCode
```

## Testing the MDS Server

It is essential that the custom activity code be tested prior to adding it to the MDS server. To accomplish this, open an interactive SAS session. Within that SAS session create a program similar to the one below. Be sure the new activity code to be tested is in the SASActivityCode directory listed.

```
/* Edit Here: Change to your install root path */
    filename acts "[SAS Install
Root]\Config\Lev1\Applications\SASRealTimeServerConfig5.4\SASActivit
yCode";
    options mautosource sasautos=(SASAUTOS,acts);

    /* Use %let statements to create input values */
    %let [my_variable]=[variable_value];


    /* Declare any input arrays or tables as global */
    %global [my_array_or_table];

    /* Use a DATA step to encode Arrays and Tables */
    data _null_;
      /* Run the encode/decode macro to setup the encode/decode
links */
      %sc_encode_decode;

        /* Setup the array/table NOTE: for testing the array/table
must be added to the output values hash object and then encoded */
        /* Set the number of rows in the array */
        sc_output_values.add(key: 'my_array', key: ., key:
'num.rows', data: '[number_of_rows]');

      /* Set the type for the array */
        sc_output_values.add(key: 'my_array', key: ., key:
'array.type', data: '[my_array_type]');

      /* Set the values for the array */
        sc_output_values.add(key: 'my_array', key: 1, key: '', data:
'[my_array_value]');

      /* Set sc_name to the name of the array and encode it */
      sc_name='my_array';
      link sc_encode_array;
    run;

    /* Execute the activity code */
    %[Activity macro name]
```

```
    /* Check the output values */
    %put [my_output_value_name]=&[my_output_value_name];

    /* Use a data step to decode Arrays and Tables */
    data _null_;
      /* Run the encode/decode macro to setup the encode/decode
links */
      %sc_encode_decode;

      /* Set sc_name to the name of the array and decode it */
      sc_name='my_array';
      link sc_decode_array;

        /* Use the sc_input_values hash object to check array/table
results */
        /* Get the number of rows in the array */
        sc_input_values.find(key: 'my_array', key: ., key:
'num.rows');
        put 'Number of rows: ' sc_data_value;

      /* Get the type for the array */
        sc_input_values.find(key: 'my_array', key: ., key:
'array.type');
        put 'Array Type: ' sc_data_value;

      /* Get the values for the array */
        sc_input_values.find(key: 'my_array', key: 1, key: '');
        put 'Value of array row 1: ' sc_data_value;
    run;
```

The following is an example of testing using the sc_variable_test sample activity.

```
/* Edit Here: Change to your install root path */
filename acts "[SAS Install
Root]\Config\Lev1\Applications\SASRealTimeServerConfig5.4\SASActivityCode";
options mautosource sasautos=(SASAUTOS,acts);

/* Use %let statements to create input values */
%let inputFloat=25000.0;
%let inputInt=7;
%let inputBoolean=1;
%let inputString=DebtConsolidation;
```

```
%let inputDate=1.81977163E8;

%let inputStringArray=String^4^String1^String2^String3^String4^;
/* Use a data step to encode Arrays and Tables */
data _null_;
      /* Run the encode/decode macro to setup the encode/decode links */
      %sc_encode_decode;

   /* Setup the array/table NOTE: for testing the array/table must be added to the
output values hash object and then encoded */
   /* Set the number of rows in the array */
   sc_output_values.add(key: 'inputStringArray', key: ., key: 'num.rows', data: '4');

      /* Set the type for the array */
   sc_output_values.add(key: 'inputStringArray', key: ., key: 'array.type', data:
'String');

      /* Set the values for the array */
   sc_output_values.add(key: 'inputStringArray', key: 1, key: '', data: 'String1');
   sc_output_values.add(key: 'inputStringArray', key: 2, key: '', data: 'String2');
   sc_output_values.add(key: 'inputStringArray', key: 3, key: '', data: 'String3');
   sc_output_values.add(key: 'inputStringArray', key: 4, key: '', data: 'String4');

      /* Encodes inputStringArray into a macro variable named inputStringArray */
      sc_name='inputStringArray';
      link sc_encode_array;
run;

/* Execute the activity code */
%sc_variable_test

/* Check the output values */
%put outputInt=&outputInt;
%put outputFloat=&outputFloat;
%put outputBoolean=&outputBoolean;
%put outputString=&outputString;
```

```
%put outputDate=&outputDate;

/* Use a data step to decode Arrays and Tables */
data _null_;
      /* Run the encode/decode macro to setup the encode/decode links */
      %sc_encode_decode;



      /* Decode outputStringArray into a macro variable named outputStringArray */
      sc_name='outputStringArray';
      link sc_decode_array;

   /* Use the sc_input_values hash object to check array/table results */
   /* Get the number of rows in the array */
   sc_input_values.find(key: 'outputStringArray', key: ., key: 'num.rows');
   put 'Number of rows: ' sc_data_value;

      /* Set the type for the array */
   sc_input_values.find(key: 'outputStringArray', key: ., key: 'array.type');
   put 'Array Type: ' sc_data_value;

      /* Set the values for the array */
   sc_input_values.find(key: 'outputStringArray', key: 1, key: '');
   put 'Value of array row 1: ' sc_data_value;
   sc_input_values.find(key: 'outputStringArray', key: 2, key: '');
   put 'Value of array row 2: ' sc_data_value;
   sc_input_values.find(key: 'outputStringArray', key: 3, key: '');
   put 'Value of array row 3: ' sc_data_value;
   sc_input_values.find(key: 'outputStringArray', key: 4, key: '');
   put 'Value of array row 4: ' sc_data_value;
run;
```

## Error Handling (SAS Activity Macro)

Two macro variables are used to indicate error conditions in SAS activity code. You must set the values for them in your custom activity code. The values are checked in SAS Real-Time Decision Manager. If the return code is non-zero, then the error message

is returned to SAS Real-Time Decision Manager; otherwise, the custom code is assumed to have executed successfully.

- **sas_connection_retCode_var** is a numeric error indicator. Any non-zero value indicates that an error has occurred.
- **sas_connection_retMsg_var** holds an error string that indicates why the error occurred. This message is written to the SAS Real-Time Decision Manager server log, but it is not returned to the user.

# Web Service Activities

## Introduction

SAS Real-Time Decision Manager functionality can be extended by adding new Web Service Activities. A Web Service Activity can invoke an external Web service to request information to be used downstream in the decision flow. For example, suppose an organization has an inventory system that exposes a Web service interface. A Web Service Activity could be created that sends a request to the inventory system to check that there is sufficient quantity of a product to extend an offer.

The Web Service Activity supports simple data types only. Arrays are not supported. Neither are binary data types.

There are issues with the current implementation of the Web Service Activity:

- The current implementation does not support array type parameters returned by a Web service. However, array type parameters are supported when calling a Web service. This is typically manifested with the following error message:

```
Activity execution in node "XXX" faulted with error code:
4,104

Xpath expression selects multiple paths which results in
ambiguity.
```

- The current implementation does not support null valued parameters when calling a Web service. In particular, null valued parameters are converted to empty strings. This may cause problems for non-string data types if parsing in the Web service is strict. For string data types, nulls are converted to empty strings.
- Security features are not supported.

The Web Service Activity uses a Web Service Connection system resource. This resource contains the URL of the Web service to invoke. When you publish a new Web Service Activity you bind it to a particular Web Service Connection system resource. Create your Web Service Connection system resource before publishing your new Web Service Activity.

# General I/O Activities

SAS Real-Time Decision Manager ships with two General I/O Activities. The end user can direct the General I/O Activities to read or write to any available database table or SAS data set. Unlike the SAS Activity and the Web Service Activity, the only motivation for adding and configuring new General I/O activities would be to tune hardware resource utilization in a clustered SAS deployment. This would be accomplished by directing the processing of the new activities to a different SAS cluster. For a discussion of SAS server clustering, see the section About SAS Connection System Resources above. A General I/O Activity uses a SAS Connection resource. This resource specifies which SAS server cluster the activity will utilize. At least one SAS Connection resource was configured when your system was installed.

# Scoring Activities

Unlike the SAS Activity and the Web Service Activity, the only motivation for adding and configuring new Scoring activities would be to tune hardware resource utilization in a clustered SAS deployment. This would be accomplished by directing the processing of the new activities to a different SAS cluster. For a discussion of SAS server clustering, see the section About SAS Connection System Resources above. A Scoring Activity uses a SAS Connection resource. This resource specifies which SAS server cluster the activity will utilize. See System Resources for more information. At least one SAS Connection Scoring resource was configured when your system was installed.

# Code Activities

The Code Activity works like the Code Node, but the Code Activity enables you to re-use the code. All of the expressions that are allowed in the Code Node are also allowed in the Code Activity (See the SAS Real-Time Decision Manager Language Reference). Currently, there is no user interface for creating a Code Activity, so you must manually create it.

The following is an example of a Code Activity in XML format:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<ActivityDefinition
      javaClassName="com.sas.analytics.ph.rt.act.code.CodeActivity"
timeout="0"
      displayName="Code Activity Test"
name="CodeActivityVariableTest"
      xmlns="http://www.sas.com/xml/analytics/rdm-1.1">
      <Description/>
      <Method displayName="sc_variable_test"
name="sc_variable_test">
            <Description>This Method Tests All the different
Variable types</Description>
            <Body>
                  <Expression>
```

```
                        MyInt = MyInt + 2;
                        MyFloat = MyFloat + 1.11;
                        MyBoolean = NOT MyBoolean;
                        MyString = Reverse(MyString);
                        MyDate = intnx('DTSECOND',MyDate,24*60*60);
                        arraySize = DIM(MyStringArray);
                        do index=1 to (arraySize / 2);
                              temp = MyStringArray[index];
                              MyStringArray[index] =
MyStringArray[arraySize - index + 1];
                              MyStringArray[arraySize - index + 1] =
temp;
                        end;
                  </Expression>
            </Body>
            <InputParameter array="false" type="Int" displayName="An
Input Integer" name="MyInt">
                  <Description>An Integer Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="Float"
displayName="An Input Float" name="MyFloat">
                  <Description>A Float Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="DateTime"
displayName="An Input Date" name="MyDate">
                  <Description>A Date Parameter in the form yyyy-MM-
dd</Description>
            </InputParameter>
            <InputParameter array="false" type="Boolean"
displayName="An Input Boolean" name="MyBoolean">
                  <Description>A Boolean (true-false)
Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="String"
displayName="An Input String" name="MyString">
                  <Description>A String Parameter</Description>
            </InputParameter>
            <InputParameter array="true" type="String"
displayName="inputStringArray" name="MyStringArray">
                  <Description>An Array of Strings</Description>
            </InputParameter>
            <InputParameter array="false" type="Int" displayName="An
Input Integer" name="arraySize">
                  <Description>An Integer Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="Int" displayName="An
Input Integer" name="index">
```

```
                  <Description>An Integer Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="String"
displayName="An Input String" name="temp">
                  <Description>A String Parameter</Description>
            </InputParameter>
            <OutputParameter array="false" type="Int"
displayName="An Output Integer" name="MyInt">
                  <Description>The result of inputInt +
2</Description>
            </OutputParameter>
            <OutputParameter array="false" type="Float"
displayName="An Output Float" name="MyFloat">
                  <Description>The result of inputFloat +
1.11</Description>
            </OutputParameter>
            <OutputParameter array="false" type="DateTime"
displayName="An Output Date" name="MyDate">
                  <Description>The result of inputDate + 1
day</Description>
            </OutputParameter>
            <OutputParameter array="false" type="Boolean"
displayName="An Output Boolean" name="MyBoolean">
                  <Description>The negation of inputBoolean - true =
false and false = true</Description>
            </OutputParameter>
            <OutputParameter array="false" type="String"
displayName="An Output String" name="MyString">
                  <Description>The result of reversing inputString -
'abc' becomes 'cba'</Description>
            </OutputParameter>
            <OutputParameter array="true" type="String"
displayName="outputStringArray" name="MyStringArray">
                  <Description>The reverse array order of
inputStringArray - String1, String2, String3 becomes String3,
String2, String1</Description>
            </OutputParameter>
      </Method>
</ActivityDefinition>
```

# Guidelines for Creating Activities

## Date Time Formats that are Supported by SAS Real-Time Decision Manager

SAS Real-Time Decision Manager I/O recognizes SAS DATETIME rather than SAS DATE.

**Note:** A SAS DATE value is a value that represents the number of days between January 1, 1960, and a specified date. A SAS DATETIME value is a value representing the number of seconds between January 1, 1960, and an hour/minute/second within a specified date.

SAS data sets can store dates as DATETIME or DATE. SAS Real-Time Decision Manager supports a single datetime data type. When datetime values are passed from SAS Real-Time Decision Manager to SAS, they are always converted into SAS DATETIME values. When these values are used to insert or update a value in a SAS data set, they will update the value as the number of seconds from January 1, 1960, rather than the number of days. If the data set column is then viewed with a DATE format for that column, then the value will display incorrectly.

SAS Real-Time Decision Manager General I/O supports only the DATETIME value. In order to insert or update a column with a DATE value a custom SAS Activity will be required. In that activity the SAS Real-Time Decision Manager DateTime object will be passed as a SAS DATETIME and then a conversion call will need to be made in the custom SAS Activity to convert the numeric to a DATE value.

## Boolean Values

Within custom SAS Activities, Boolean values must be represented as the numerics 0 and 1, as opposed to True and False.

## General I/O Write and SAS Data Sets

SAS data sets do not support concurrent updates; therefore, a locking error could occur when trying to use the General I/O activity to update or insert into a SAS data set. If concurrent writes are required, then use another data base. If SAS data sets are required, then the polling server that is used to handle the updates would have the MAX SESSIONS option set to 1 indicating a single session. This will queue all of the WRITE activities to the SAS data set and execute them one at a time.

**Note:** This action is for WRITE only; concurrent READs are fully supported.

If the data set is opened in SAS while the SAS Real-Time Decision Manager Connection server is processing requests, then locking errors will occur because SAS locks the table when it is opened. It is recommended not to open a SAS data set in an interactive SAS session while SAS Real-Time Decision Manager is using the SAS data set.

CHAPTER 4

# Data Collection for Performance Analysis



## The SAS Real-Time Decision Manager User Log

The intended purpose of the user log is for the administrator to get information on how specific events flow through the engine. It is only meant to be turned on for short periods of time. It can also be used to debug specific events. The user log should not be used in the production environment as it has a significant performance impact.

The SAS Connection System Resource that is used by the user log is specified in the rtdm_config.properties file as sas.rdm.performance.monitor.system.resource. This value is set as part of the configuration process. This resource should reference a JMS Request Queue that is being handled by a single Message Queue Polling Server process. You must use the Audit Log SAS Connection System resource because it has the same constraints as those of a single Message Queue Polling server process.

### What the SAS Real-Time Decision Manager User Log Contains

The user log contains several XML documents. Each document has a top level element called TestOutput. The XML document contains all of the following information that is needed for a single RTDM Webservice Event call:

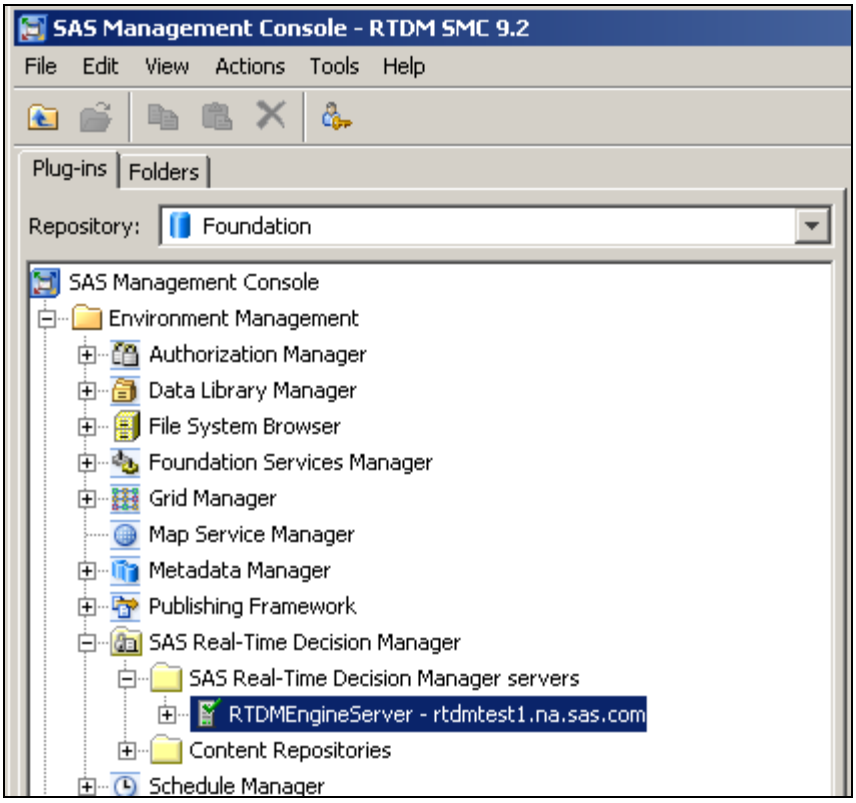
- the event request data
- the values of the process variables before executing each activity
- the values of the process variables after executing each activity
- the path that the event traveled through the flow

### Location of the User Log

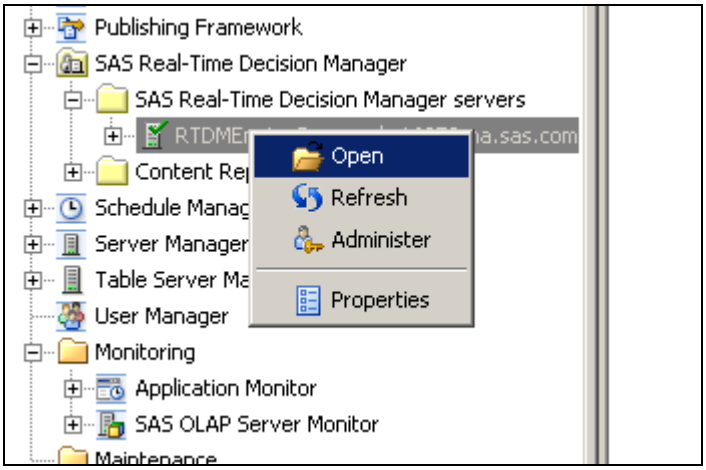
The location of the user log is set during the installation and configuration process of SAS Real-Time Decision Manager. The default directory is <*CONFIG LEV1 DIRECTORY*> Applications\SASRealTimeServerConfig5.4\Logs\UserEvents.log; where <*CONFIG LEV1 DIRECTORY*> is the root directory for the particular installation.

To enable the user log, follow these steps:

1. Open the SAS Management Console.
2. Expand **Real-Time Decision Manager** and **SAS Real-Time Decision Manager Servers**.

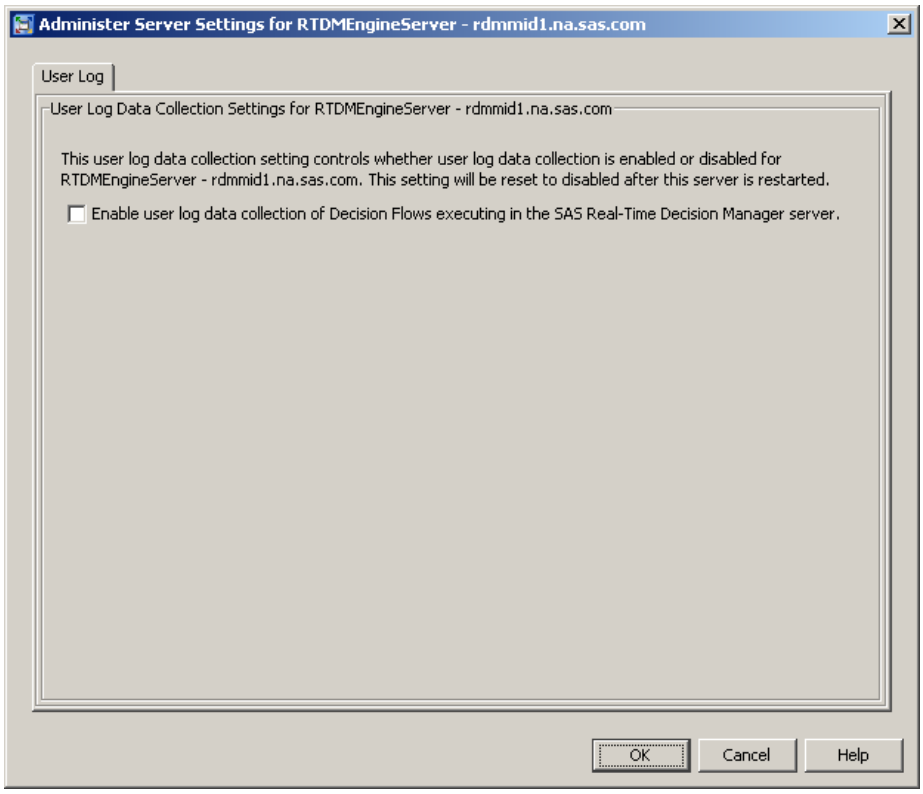



3. Right-click the system that you wish to collect performance data for, and select **Administer**.




4. On the **User Log** tab, check the check box for **Enable user log data collection of decision flows executing in the SAS Real-Time Decision Manager server**.

   **Note:** Enabling data collection affects performance. To disable data collection to a log, remember to clear this box.


Administer Server Settings for RTDMEngineServer - rdmmid1.na.sas.com
User Log
User Log Data Collection Settings for RTDMEngineServer - rdmmid1.na.sas.com
This user log data collection setting controls whether user log data collection is enabled or disabled for RTDMEngineServer - rdmmid1.na.sas.com. This setting will be reset to disabled after this server is restarted.
Enable user log data collection of Decision Flows executing in the SAS Real-Time Decision Manager server.
OK
Cancel
Help

CHAPTER 5

# Audit Logging



## Overview of the Audit Logger

The audit logger collects information about events that occur in the SAS Real-Time Decision Manager engine, and records the data in SAS data sets (see Tables of Audit Logging Events). Events are logged from the SAS Real-Time Decision Manager engine server and from the SAS Real-Time Decision Manager Plug-in for SAS Management Console.

These engine events are logged to SAS data sets:

- cached flow
- cached global variables
- engine stop
- flow activate
- flow deactivate

When a global variable value is changed in the SAS Real-Time Decision Manager plug-in, the cached global variables and cached flows are logged.

### Terms That Are Used in Audit Logging

The following data items are common to all events:

- **GUID** a globally unique ID that is used as the primary key in order to link data in multiple tables.
- **Host Name** used to group several events from an engine server and from the SAS Real-Time Decision Manager plug-in.
- **Object Name** a column that contains the name of a flow or of a global variable.
- **Object Type** one of the following: an engine, a flow, or a global variable.
- **Operation** the type of event that is being logged, such as Cached, Activate, Deactivate, Stop.

- **Timestamp** a sequence of characters that denote the date and time at which a certain event occurred. Because the WebSphere MQ listener polls multiple SAS processes in order to pull messages from the Message Queue Polling Servers, the timestamp is used to recover the order of messages.

# Setting Up the Audit Logging Functionality

The components for the audit logger are configured during the installation and configuration of SAS Real-Time Decision Manager. A message queue polling server is created specifically for the audit logger. The message queue polling server uses a designated system resource called $AuditLogSASConnection.

The SAS data tables (that are required for audit logging) are created when the first operation is performed. If the first operation is caching a flow by the SAS Real-Time Decision Manager engine, then the table that contains that flow caching event is created. When SAS Real-Time Decision Manager is installed, no flows are active; therefore, no entries exist that need to be cached. When a flow is activated, the corresponding event is logged. The engine then caches the flow, which is another event that is logged.

## Tables of Audit Logging Events

Audit logging events are recorded in these four SAS data sets that are located in the [*install dir*]\Applications\SASRealTimeServerConfig5.3\Data\CONNLIB directory:

- AuditLog
- AuditLogFlows
- AuditLogGlobals
- AuditLogGlobalValues

The AuditLogFlows table is related to the AuditLog table via the key GUID. The relationship of AuditLog to AuditLogFlows is one-to-many.

The AuditLogGlobals table is also related to the AuditLog table via the key GUID. The relationship of AuditLog to AuditLogGlobals is one-to-many.

The AuditLogGlobalValues table is related to the AuditLogGlobals table via the key GUID + Name. The relationship of AuditLogGlobals to AuditLogGlobalValues is one-to-many.

## Data That Is Logged for Cached Global Variable Events

The following information is logged for each Cached Global Variables event:

- GUID
- Name
- Type
- IsArray
- Value

- Index

### Data that is Logged for Cached Flows Events

The following information is logged for each Cached Flows event:

- GUID
- Name
- Type

## Audit Log Locations

The following table summarizes the location of log files generated by the various SAS Real-Time Decision Manager components.

| Log Locations | |
|---|---|
| **Component** | **Default Location of logs** |
| *SAS Tier* | *<SAS Config> is the location for installing the configuration for SAS e.g. D:/SAS/Config/Lev1*<br>*yyyy-mm-dd is the date time of creation of the log file.*<br>*nnnn is a 4 digit number to differentiate between logs created by multiple processes that perform the same function.*<br>*N.N is the major and minor version number of SAS RTDM* |
| Metadata Server | <SAS Config>/SASMeta/MetadataServer/Logs/SAS_Meta_MetadataServer_yyyy-mm-dd_nnnn.log |
| Object Spawner | <SAS Config>/ObjectSpawner/Logs/ObjectSpawner_ yyyy-mm-dd_nnnn.log |
| SAS RTDM Single Data Step Server | <SAS Config>/Applications/SASRealTimeServerConfigN.N/Logs/SingleDataStep_yyyy-mm-dd_nnnn.log |
| SAS RTDM Multiple Data Step Server | <SAS Config>/Applications/SASRealTimeServerConfigN.N/Logs/MultipleDataStep_yyyy-mm-dd_nnnn.log |
| SAS RTDM Audit Log | <SAS Config>/Applications/SASRealTimeServerConfigN.N/Logs/AuditLog_yyyy-mm-dd_nnnn.log |
| SAS RTDM Model Update Service | <SAS Config>/Applications/SASRealTimeServerConfigN.N/Logs/ModelUpdate_yyyy-mm-dd_nnnn.log |
| *Mid-tier* | |
| Remote Services | <SAS Config>/Web/Logs/RemoteServices.log |
| SAS RTDM Engine Server | <SAS Config>/Web/Logs/SASRealTimeEngineServerN.N.log |
| SAS RTDM Design Server | <SAS Config>/Web/Logs/SASRealTimeDesignServerN.N.log |
| *IBM Websphere MQ* | *<MQ Base> is the location where MQ is installed, e.g. C:/Progam Files (x86)/IBM/Websphere MQ*<br>*<Queue Manager Name> is the name of the queue manager. The RTDM install uses QM_QMANAGER by default. n is a number. Typically there are 3 log files 1-3.* |
| Client | <MQBase>/errors/AMQERR0n.log |
| Queue Manager | <MQBase>/Qmgrs/<Queue Manager Name>/errors/ AMQERR0n.log |
| *IBM Websphere Application Server* | *<WAS Base> is the location where Websphere Application Server is installed, e.g. D:/IBM/Websphere/AppServer.*<br>*<profile name> is the name of the server profile. E.g. the profile would be called SAS<node* |

| | |
|---|---|
| | *name> 01 if*<br>*<node name> is the name of a node.*<br>*<server name> is the name of the server on which RTDM web applications are deployed. By default, both the RTDM Engine and RTDM Design Server are deployed on one called SASServer7.*<br>*There are several logs for each server. Details are available in the WAS documentation. For debugging RTDM errors, the most significant ones are called SystemOut.log and SystemErr.log.* |
| SystemOut | <WAS Base>/profiles/<profile name>/logs/<server name>/SystemOut.log |
| SystemErr | <WAS Base>/profiles/<profile name>/logs/<server name>/SystemErr.log |

CHAPTER 6

# Web Services Integration



The SAS Real-Time Decision Manager Engine Server is seen externally as a Web service endpoint. External applications request decisions by sending Web service requests to SAS Real-Time Decision Manager.

When the SAS Real-Time Decision Manager Web service endpoint is triggered by a Simple Object Access Protocol (SOAP) Event request, the Web service maps the incoming request to a SAS Real-Time Decision Manager Event object and passes it to the run-time engine for processing. After the run-time engine has completed its processing, a SOAP response is serialized back to the invoking client. One-way Event operations are also supported, which do not follow the common request and response message exchange pattern just described. In this case, a client sends a request and does not expect a response.

Specifically, SAS Real-Time Decision Manager supports SOAP document-style encoding, also known as document-literal or message-style encoding. Of the three most popular SOAP encoding styles, SOAP RPC, SOAP RPC-literal, and document-literal, the document-literal style has the least overhead and highest performance.

The variables in the SOAP messages are accessed by name and the order of declaration is not significant. In particular, the variables in the SOAP messages are independent of the order of the variables defined in the request and reply message sections of the event definition.

Consequently, the variables in the request SOAP message do not have to be in the same order, as defined in the request message section of the event definition. Also, the SOAP client cannot rely on the variables in the reply SOAP message to be in any particular order, such as the order as defined in the response message section of the event definition.

## Web Service Definition Language

You can retrieve the Web Service Definition Language (WSDL) from the SAS Real-Time Decision Manager Web service endpoint as follows:

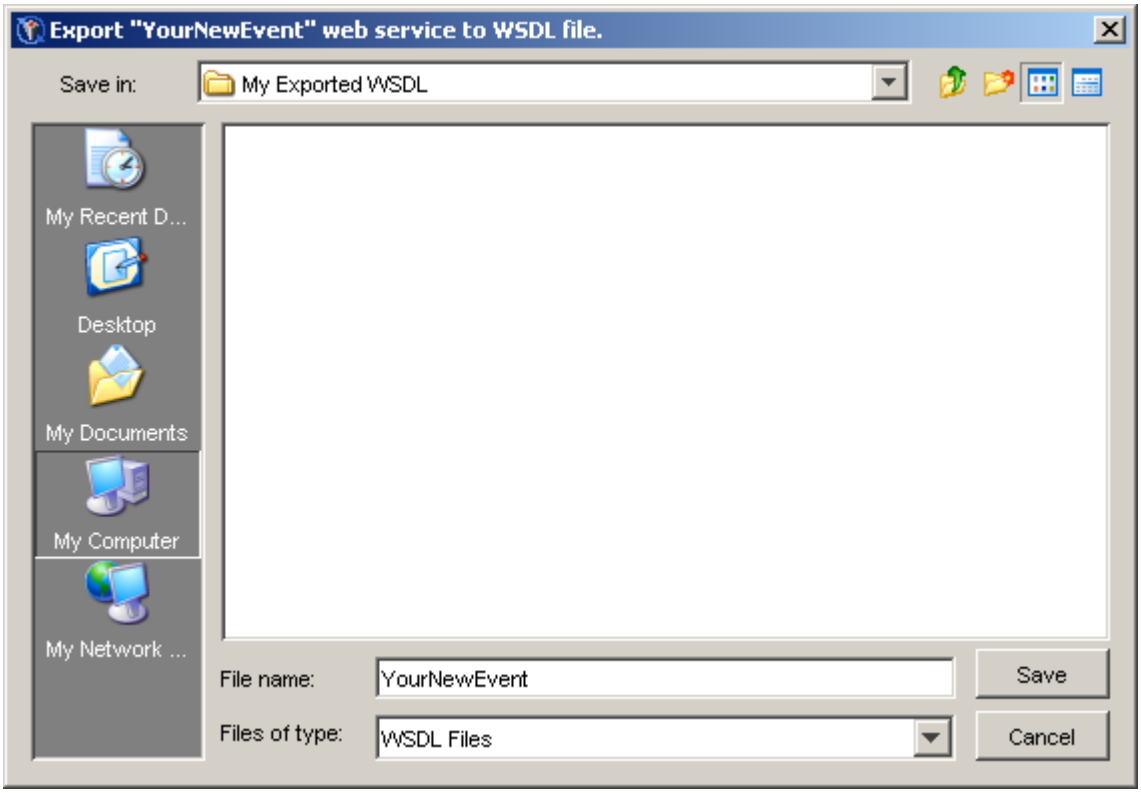
```
http://hostname:WC_Defaulthost_port/RDM/Event?wsdl
```

*Hostname* is the URL of the server on which the SAS Real-Time Decision Manager engine is deployed. See the section Record the Port Values and Administration Servlet URL for the WC Default host port number.

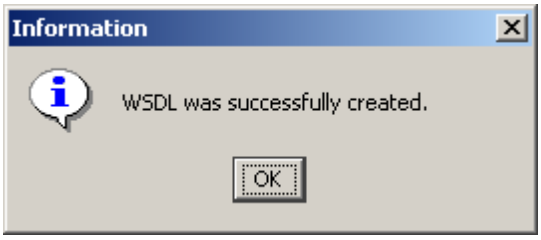
The WSDL returned is a general purpose WSDL that will work with any event. It supports zero to more elements of any simple, non-binary type.

You can create a specific WSDL for a given SAS Real-Time Decision Manager event by invoking an export process as follows:

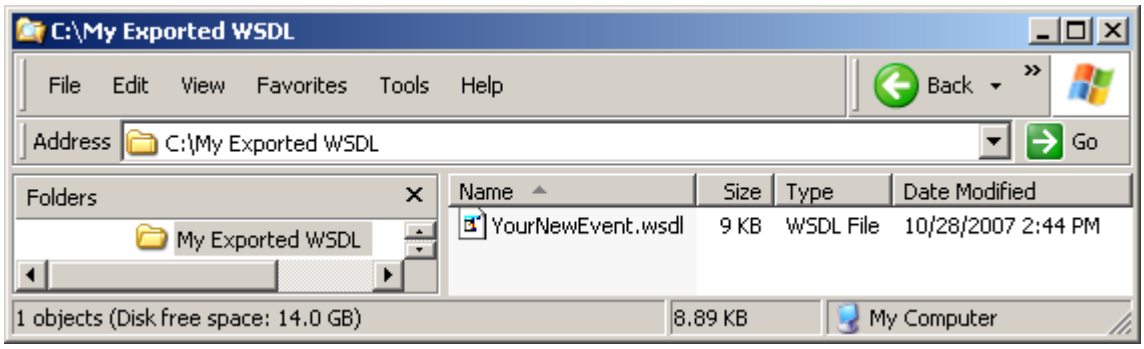
1. Open SAS Management Console.
2. On the **Folders** tab, select **System, Applications, SAS Real-Time Decision Manager 5.3**.
3. Navigate to the SAS Real-Time Decision Manager repository for which you want to generate a WSDL.
4. Right-click the Web Service event in the repository and **select Export WSDL**.
5. Modify the default address for your environment. A sample address is: `http://localhost:9086/RTDM/Event`. The address is determined during the installation of your software.
6. Browse to a location to store and name the WSDL file.




7. Click Save.




You can browse your file system to verify that the new WSDL exists.




## Sample Web Service Request

After creating an event and mapping that event to a decision flow, you can deploy the flow to a running instance of the SAS Real-Time Decision Manager engine server. Once activated, it can be invoked by a Web service client. Here is a sample instance of a SOAP request that calls an event named “CustomerCall”.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <env:Header/>
    <env:Body>
       <rdm:Event xmlns:rdm="http://www.sas.com/xml/analytics/rdm-1.1"
name="CustomerCall">
          <rdm:Header>
             <rdm:Identity>John Smith</rdm:Identity>
            <rdm:ClientTimeZoneID>America/New_York</ClientTimeZoneID>
          </rdm:Header>
          <rdm:Body>
             <rdm:Data name="CustomerID">
                <rdm:String>
                   <rdm:Val>001</rdm:Val>
                </rdm:String>
             </rdm:Data>
             <rdm:Data name="Amount">
                <rdm:Float>
                   <rdm:Val>25000.0</rdm:Val>
                </rdm:Float>
             </rdm:Data>
             <rdm:Data name="Mood">
                <rdm:String>
                   <rdm:Val>Good</rdm:Val>
                </rdm:String>
             </rdm:Data>
```

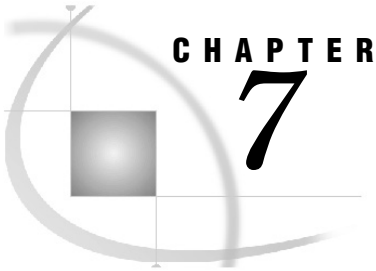
```
            <rdm:Data name="SigEvent">
               <rdm:String>
                  <rdm:Val>NewBaby</rdm:Val>
               </rdm:String>
            </rdm:Data>
         </rdm:Body>
      </rdm:Event>
   </env:Body>
</env:Envelope>
```

`ClientTimeZoneID` is a required tag in the Real-Time Decision Manager header. Time Zone names from the public domain time zone (TZ) database are accepted. Here is a Web site listing time zone information from the TZ database: http://home.tiscali.nl/~t876506/TZworld.html#nam.

Every Web service stack has client tools that can be used to generate stubs and helper classes that call particular Web services. These toolsets will take the desired Web service's WSDL as input and generate the stubs and helper classes as output. Clients can be plain Java or .Net applications, or in a J2EE world, they can be J2EE application clients or J2EE Web applications themselves.

The variables in the SOAP messages are accessed by name. The order of declaration is not significant. In particular, the variables in the SOAP messages are independent of the order of the variables defined in the request and reply message sections of the event definition.

Consequently, the variables in the request SOAP message do not have to be in the same order as defined in the request message section of the event definition. Also, the SOAP client cannot rely on the variables in the reply SOAP message to be in any particular order, like the order as defined in the response message section of the event definition.



CHAPTER 7

# Integration with SAS Model Manager



## About SAS Model Manager

SAS Model Manager, licensed separately, may be integrated with SAS Real-Time Decision Manager to provide an end-to-end solution for managing and deploying analytical models into real-time operational environments.

SAS Model Manager 2.2 is the required version for use with SAS Real-Time Decision Manager.

Refer to the SAS Model Manager documentation for instructions on how to use that product. This section describes the integration and interoperability between SAS Real-Time Decision Manager and SAS Model Manager.

SAS Real-Time Decision Manager ships with an activity called Scoring. When integrated with SAS Model Manager, the Scoring activity provides several features:

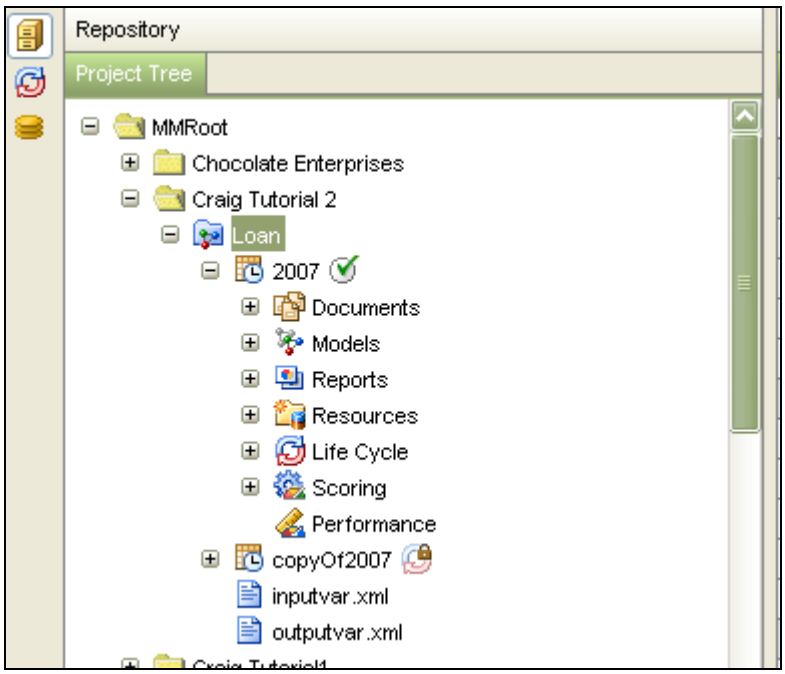
- In the development environment, the Scoring activity allows a user to choose any of the scoring projects that have been published to SAS Real-Time Decision Manager by SAS Model Manager. The Scoring activity may be added to a decision flow in multiple places, allowing multiple models to be included in a single decision flow.
- In the production environment, whenever a decision flow that includes Scoring activities is executed, those activities execute the referenced score code.
- SAS Real-Time Decision Manager does not reference models directly. Rather, it references scoring projects, which in turn contain models. At runtime, the champion model within the scoring project is used. This indirection allows new versions of models to be hot-deployed without redeploying the decision flows that reference them.

## Publish a Scoring Project

The installation and configuration of Model Manager creates the directory structure in SAS Management Console for housing scoring projects. The default location (on the **Folders** tab in SAS Management Console) is Shared Data/Model Manager/Publish/Projects.

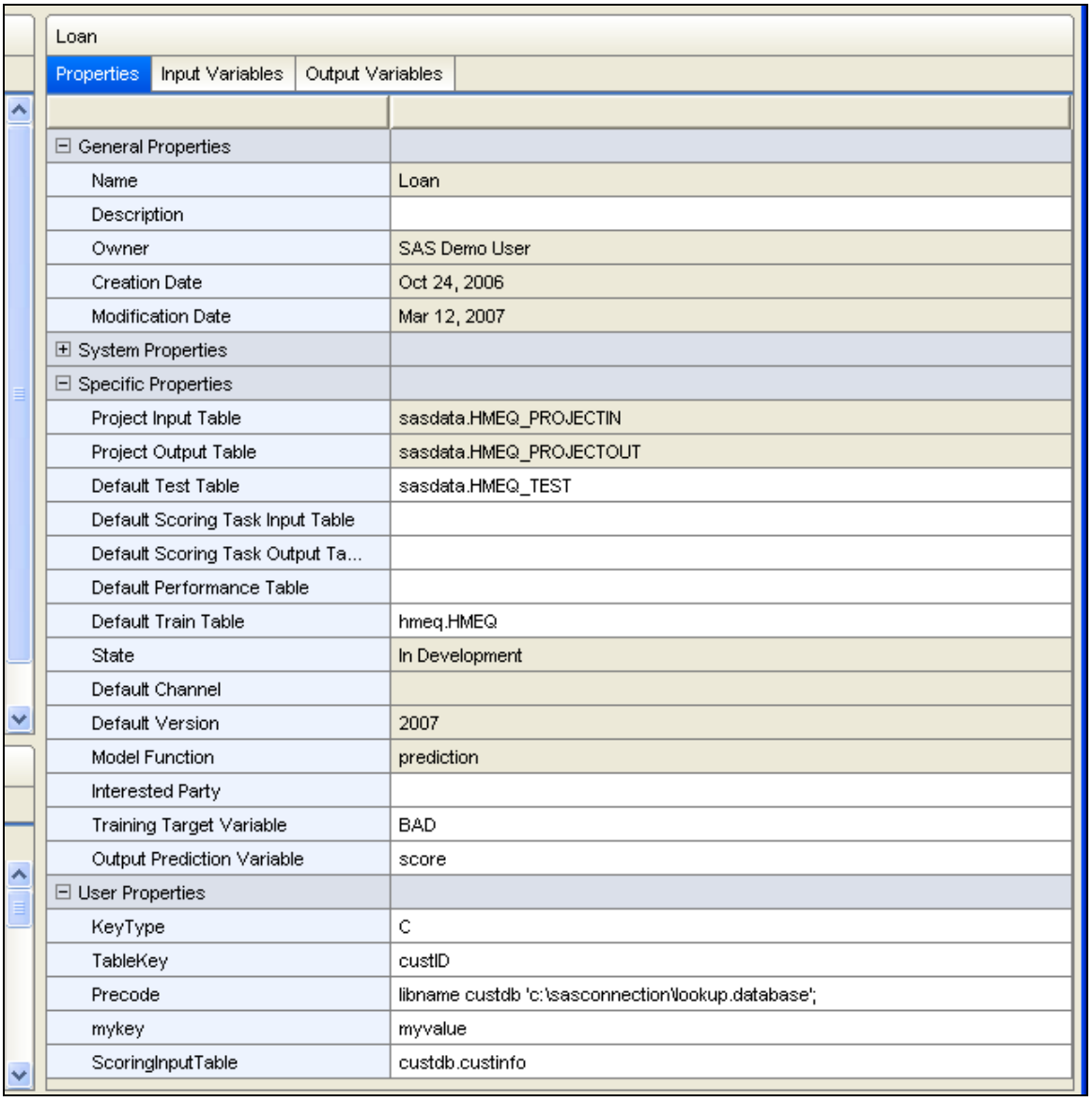
To publish a scoring project to SAS Real-Time Decision Manager, follow these steps:

1. Open Model Manager and navigate to the scoring project you wish to publish. The example uses the Loan project:




2. In the User Properties, add the following name-value pairs to the project. These are used by the score code to read predictors from the scoring input data set.

| | |
|---|---|
| **KeyType** | Data type of the primary key (C for Character in the example below or N for Numeric) |
| **TableKey** | Primary key (custID in the example below) |
| **PreCode** | Pre-code can contain any SAS code that should be executed prior to score code execution. In the example below, it contains the libref of the library containing the scoring input data set (libname custdb 'c:\sasconnection\lookup.database';). Another example would be to add an option statement to include a format search path if user-defined SAS formats were used by the scoring code. |
| **ScoringInputTable** | Data set name or relational table name (custdb.custinfo in the example below) |



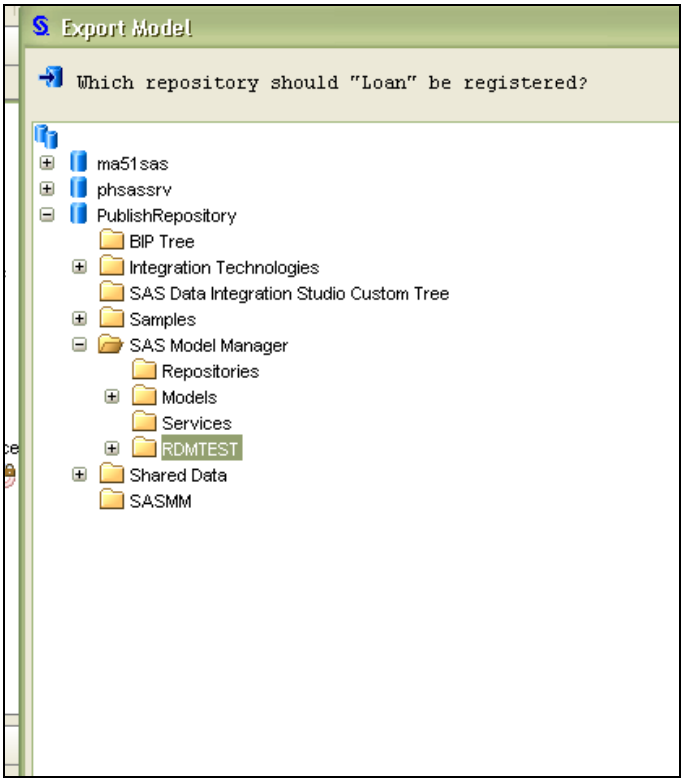
Loan

Properties | Input Variables | Output Variables

| | |
|---|---|
| General Properties | |
| Name | Loan |
| Description | |
| Owner | SAS Demo User |
| Creation Date | Oct 24, 2006 |
| Modification Date | Mar 12, 2007 |
| System Properties | |
| Specific Properties | |
| Project Input Table | sasdata.HMEQ_PROJECTIN |
| Project Output Table | sasdata.HMEQ_PROJECTOUT |
| Default Test Table | sasdata.HMEQ_TEST |
| Default Scoring Task Input Table | |
| Default Scoring Task Output Ta... | |
| Default Performance Table | |
| Default Train Table | hmeq.HMEQ |
| State | In Development |
| Default Channel | |
| Default Version | 2007 |
| Model Function | prediction |
| Interested Party | |
| Training Target Variable | BAD |
| Output Prediction Variable | score |
| User Properties | |
| KeyType | C |
| TableKey | custID |
| Precode | libname custdb 'c:\sasconnection\lookup.database'; |
| mykey | myvalue |
| ScoringInputTable | custdb.custinfo |

Other user-defined attributes can be included in addition to the four described above. For example, you could add CampaignCode=1234. The additional name/value pairs can be helpful to business users in searching for models from SAS Customer Intelligence Studio.

3. Export the project (Loan in the example) to the publish folder (RDMTEST) in the BIP tree. Note that two levels of export are supported by SAS Model Manager, project-level and model-level. Always export from the project-level.




The project information and files have now been transferred to the Metadata Server and the project is ready to be extracted by the Model Update Service. The project information includes the score code, input, and output variables, which are needed for scoring, as well as information about the project and champion model that can be presented in SAS Customer Intelligence Studio 5.4.

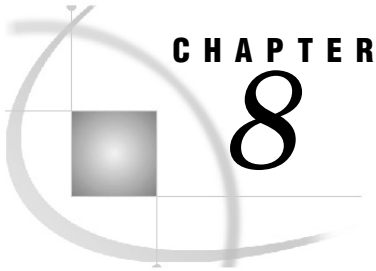
## Model Update Service

The Model Update Service distributes scoring projects that are published by SAS Model Manager to the SAS servers in a cluster. The Model Update Service periodically checks for newly published scoring projects and, if found, distributes the champion model score code from the projects to the SAS servers within the cluster. The Model Update service runs as a polling server.

## Best Practices

As a best practice, one OMR folder for publishing models should be created for each development, test, and production SAS Real-Time Decision Manager environment in your deployment. A scoring project should be published to the development folder first and tested in the SAS Real-Time Decision Manager development environment.

When a decision flow that references that scoring project is promoted to a test or production environment, then the scoring project should be published to the test or production folder at that time. Using this practice, the same testing, approval, and promotion policies that are applied to decision flows can be applied to scoring projects.



CHAPTER 8

# Integration with SAS Data Surveyor for Clickstream Data



## About SAS Data Surveyor for Clickstream Data

*Clickstream* is a term used to describe the data that is collected from users as they access Web sites through various electronic devices. Clickstream data includes the stream of user activity stored in a log. Clickstream data can be collected and stored in a variety of ways. The SAS Data Surveyor for Clickstream Data provides the capability to process this data into meaningful results.

Integration of SAS Data Surveyor for Clickstream Data with SAS Real-Time Decision Manager allows for real-time campaign content to be presented to the Web site visitor based on information specific to the visitor's session. Any subsequent activity as a result of user actions taken upon the presented content will be tracked. This can help with determining the success of campaigns and analyzing customers' responses to different types of content presented within a campaign.

## Real-Time Behavior Tracking and Analysis

You can combine the functionality provided by SAS Data Surveyor for Clickstream Data and SAS Real-Time Decision Manager. This combination enables you to dynamically update Web site content targeted to the customer in real time. Therefore, you can record and track any response as a result of activity on the modified page. This approach provides a way to dynamically present targeted content to a customer. It also facilitates establishing the effectiveness of the campaign.

The SAS page tag functionality passes session information through an asynchronous request to the SAS Real-Time Decision Manager Web service, which responds with a targeted response. The response contains information about a treatment that should be displayed on the current Web site. For example, the treatment can identify an image contained in the content management system that should be displayed on the Web site. The SAS page tag functionality then updates the Web page source to display the appropriate content. If the customer then clicks on this treatment (which is typically a link), information about the campaign that generated the treatment is recorded in the SAS page tag log and later processed as part of the ETL.

# For More Information

See SAS Data Surveyor for Clickstream Data User's Guide for detailed information about configuration and use of the combined functionality provided by SAS Data Surveyor for Clickstream Data and SAS Real-Time Decision Manager.

CHAPTER 9

# Installation and Configuration

Before installing SAS Real-Time Decision Manager, work with your on-site SAS support personnel to determine the hardware, network, and software topology required to meet your throughput and response time needs.

At a minimum, one development and one production environment should be installed. One or more test environments may be installed, depending upon your organization's testing policies. Decision flows may be unit tested in the development environment. A test environment is used to test decision flows in an environment similar to production. The only differences between the test and production environments are:

- The test environment will not be connected to "live" channels or customer-facing systems.
- The production environment might have more hardware and network resources allocated to it.

The development environment is typically not clustered. The production environment might utilize a clustered middle tier, SAS tier, database tier, and WebSphere MQ, depending on performance requirements.

The middle tier runs inside the WebSphere Application Server container. The IBM WebSphere Application Server provides a set of services for managing ecommerce applications using the Java 2 Enterprise Edition (J2EE) standards.

IBM WebSphere MQ is a message-oriented middleware product that provides communications between the SAS Real-Time Decision Manager middle tier and SAS MVA server tier. IBM WebSphere MQ 7.0.0.1 must be installed and configured prior to SAS Real-Time Decision Manager. Contact your IBM representative or see http://www-306.ibm.com/software/integration/wmq/ for more information about IBM WebSphere MQ.

For details about IBM WebSphere Application Server, refer to www.ibm.com.

For the development environment, the WebSphere Application Server (only) deployment is recommended. This is a non-clustered deployment. For the production environment, a clustered WebSphere Application Server and Deployment Manager with HTTP Server is recommended for applications requiring high availability or horizontal scalability. This is referred to as "WebSphere Application Server with Network Deployment" in the IBM literature.

# Installation Overview

There are two separate installation procedures, one for the SAS Real-Time Decision Manager development environment, and another installation for the production (or test) environment.

## Overview: Installing a Development Environment

The following general steps are used to install and configure a development environment. For detailed steps, see Install and Configure a Development Environment in this guide.

1. Install WebSphere Application Server or gain access to an existing installation.
2. Configure the WebSphere environment:
    a. Create a profile for a single application server.

   b. Record the port numbers and administration servlet URL (needed by the SAS Real-Time Decision Manager installation).
   c. Define a new object cache instance.
3. Install WebSphere MQ.
4. Set up WebSphere MQ manually, or use the SAS Deployment Wizard to set up WebSphere MQ automatically.
5. Use the SAS Deployment Wizard to install the SAS Real-Time Decision Manager development environment.
6. Start the SAS Object Spawner.
7. Start Remote Services. Deploy the SAS Real-Time Decision Manager design server EAR into the WebSphere environment.
8. Start the SAS Real-Time Decision Manager design server.
9. Verify the SAS Real-Time Decision Manager design server deployment.
10. Deploy the SAS Real-Time Decision Manager engine server EAR into the WebSphere environment.
11. Start the SAS Object Spawner.
12. Start the SAS Real-Time Decision Manager engine server.

Verify the SAS Real-Time Decision Manager engine server deployment.

## Overview: Installing a Production (or Test) Environment

The following general steps are used to install and configure a production environment. For details, see Install and Configure a Production Environment in this guide.

1. Install WebSphere Application Server or gain access to an existing installation.
2. Configure the WebSphere environment:
   a. Configure the WebSphere Application Server with Network Deployment.
   b. Install an IBM HTTP Server.
   c. Record the port numbers and administration servlet URL (needed by the SAS Real-Time Decision Manager installation).
   d. Create a new cluster.
   e. Add members to the cluster.
   f. Create a Web server definition.
   g. Define a new object cache instance.
   h. Create and configure a new virtual host.
3. Install WebSphere MQ.
4. Set up WebSphere MQ manually, or use the SAS Deployment Wizard to set up WebSphere MQ automatically.
5. Use the SAS Deployment Wizard to install the SAS Real-Time Decision Manager production environment.
6. Manually deploy the SAS Real-Time Decision Manager engine server EAR into the clustered WebSphere environment.
7. Create the HTTP Server Plug-in for distributing requests in the cluster.
8. Start the SAS Object Spawner.
9. Start the Remote Services.
10. Start the SAS Real-Time Decision Manager engine server cluster.

Verify the SAS Real-Time Decision Manager engine server deployment.

# Install and Configure a Development Environment

## Install WebSphere Application Server

If you plan to perform a manual installation of WebSphere, then install IBM WebSphere Application Server 6.1 with fix pack 21, and define a profile for a single application server. For information on how to install WebSphere, refer to your IBM vendor. Also see:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/tpro_instancesdmgr.html

## Create a Profile for a Single Application Server

In most cases, when you installed your WebSphere application server, you defined your profile at that time. In the event that you have not defined your profile, brief instructions are included here to define a single application server profile.

1. Start your WebSphere Application Server if it is not already started
2. Start the profile creation wizard.
3. Select Next.
4. Select the radio button **Create an application server profile**
5. Select Next.
6. Enter a profile name: _____
7. Select your profile directory:_____
8. Select your Node Name:_____
9. Select your Host Name:_____

## Record the Port Values and Administration Servlet URL

If you followed the instructions above to install WebSphere Application Server and create a single application server profile, then use the first scenario below. If you are working with a previously installed WebSphere Application Server, then find out from your WebSphere administrator which of the following four WebSphere Application Server configurations exist at your site:

- Application Server
- Application Server and Deployment Manager
- Clustered Application Server and Deployment Manager
- Clustered Application Server and Deployment Manager with HTTP Server

Follow the instructions for your particular WebSphere Application Server configuration.

## Application Server

To obtain the port numbers required by the SAS Real-Time Decision Manager installation:

1. Open the WebSphere Administrative Console.
2. Select Servers.
3. Select Application Servers.
4. Select your application server's name.
5. Select Ports under Communications.

Record the values for the following ports. The values will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 2809 | |
| SOAP_CONNECTOR_ADDRESS | 8880 | |
| WC_DEFAULTHOST | 9080 | |
| ORB_LISTENER_ADDRESS | 9100 | |

Substitute the appropriate values (in bold) into the following URL and record it. It will be needed during the SAS Real-Time Decision Manager installation process.

http://application-server-name:WC-defaulthost-port/RTDM/Event

## Application Server and Deployment Manager

To obtain the port numbers required by the SAS Real-Time Decision Manager installation:

1. Open the WebSphere Administrative Console.
2. Select System Administration.
3. Select Deployment Manager.
4. Select Ports.

Record the values for the following ports. The values will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 9809 | |
| SOAP_CONNECTOR_ADDRESS | 8879 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9101 | |

Substitute the appropriate values (in bold) into the following URL and record it. The URL will be needed during the SAS Real-Time Decision Manager installation process.

http://application-server-name:WC-defaulthost-port/RTDM/Event

### Clustered Application Server and Deployment Manager

To obtain the port numbers required by the SAS Real-Time Decision Manager install:

1. Open the WebSphere Administrative Console.
2. Select **System Administration.**
3. Select **Deployment Manager.**
4. Select **Ports.**

Record the values for the following ports. They will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 9809 | |
| SOAP_CONNECTOR_ADDRESS | 8879 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9101 | |

In the following URL, substitute the appropriate values for the placeholders (in bold) and record the URL. It will be needed during the SAS Real-Time Decision Manager installation process.

http://application-server-name:WC-defaulthost-port/RDTM/Event

### Clustered Application Server and Deployment Manager with HTTP Server

To obtain the port numbers required by the SAS Real-Time Decision Manager install:

1. Open the WebSphere Administrative Console.
2. Select **System Administration.**
3. Select **Deployment Manager.**
4. Select **Ports.**

Record the values for the following ports. The values will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 9809 | |
| SOAP_CONNECTOR_ADDRESS | 8879 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9101 | |

Substitute the appropriate values (in bold) into the following URL and record it. It will be needed during the SAS Real-Time Decision Manager installation process.

http://HTTP-server-name/RTDM/Event

## Define a New Object Cache Instance

Start the WebSphere Administration Console and navigate to **Resources→cacheinstances→ObjectCache Instances** and click **New**.

1. Select the **Object Cache Scope**.
2. Set the scope of the object cache instance by highlighting the server scope and selecting the radio button.
3. Select **New.**
4. Enter the Object Cache Instance Properties.
   **Name:** RTDMCache
   **JNDI Name**: cache/RTDM
   **Cache Size:** 12000
   **Default Priority**: 1
5. Select **Apply**.
6. Select **Save**.
7. Click the check box beside **Synchronize changes with Nodes**

*Make a note of the cache name and the JNDI name*.

## Install WebSphere MQ

Install WebSphere MQ 7.0.0.1 by following the instructions provided by IBM for installing on Solaris:

http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.amqdac.doc/sq10120_.htm

Separate MQ client and server installations are required.

WebSphere MQ can either be set up manually (see the following instructions), or automatically (using the SAS Deployment Wizard).

Follow these steps to manually set up your WebSphere MQ software queue manager, queues, channel, and listener:

From a UNIX command line, follow these steps by issuing the following commands:

a. Create your SAS Real-Time Decision Manager Server Queue Manager:
```
crtmqm -q venus.queue.manager
```

b. Start your SAS Real-Time Decision Manager Server Queue Manager:
```
lstrmqm
```

c. Run the MQSC for Websphere MQ:
```
runmqsc
define qlocal (request.queue)
define qlocal ()
define listener (LISTENER1) trptype (tcp) control (QMGR) port
(1414)
start listener (LISTENER1)
define channel (CHANNEL1) chltype (svrconn) trptype (tcp)
start channel (CHANNEL1)

end
```

## Use the SAS Deployment Wizard to Install the SAS Real-Time Decision Manager Development Environment

### *Run the Installation Program*

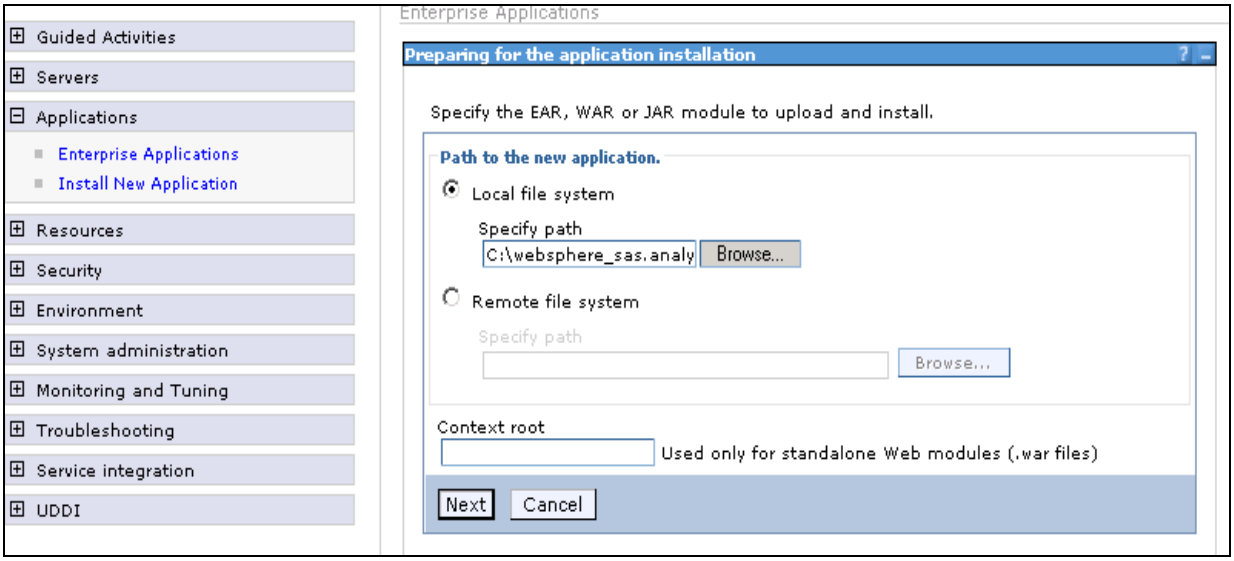
SAS support personnel will build a custom plan file for your specific SAS Real-Time Decision Manager installation. This planning document will be used as input to the SAS Deployment Wizard.

As the SAS Deployment Wizard executes, answer the prompts to provide the requested information about your installation environment.

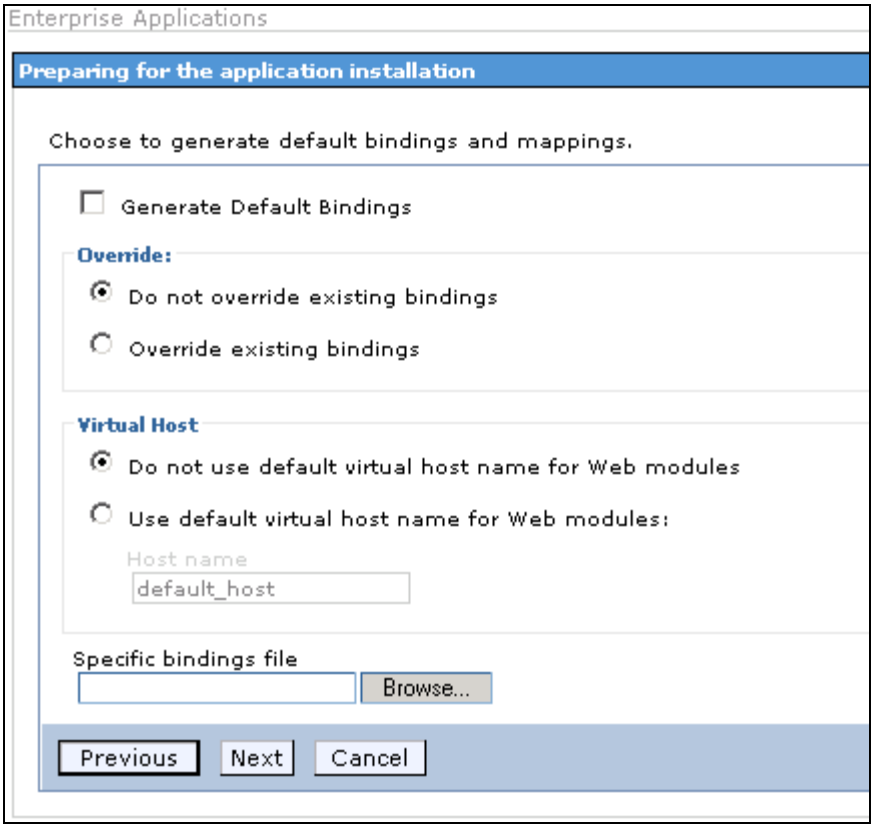
## Deploy the SAS Real-Time Decision Manager Design Server EAR into the WebSphere Environment

Start the WebSphere Application Server Administration Console.

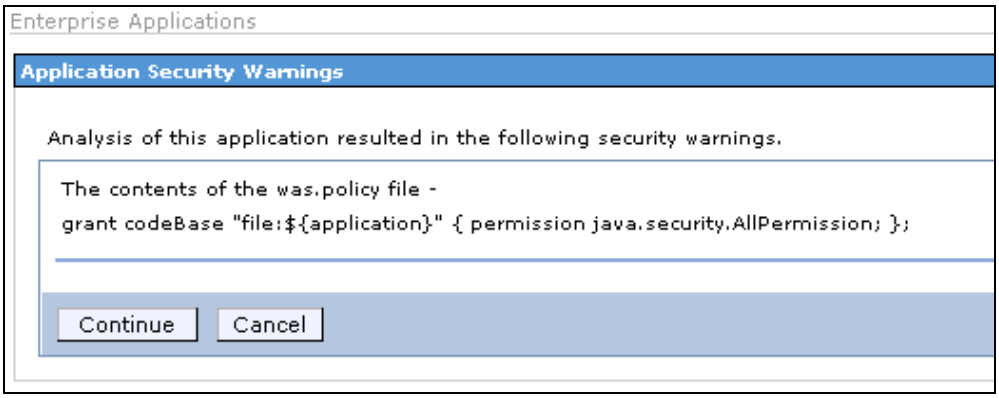
1. Select **Install New Application**.
2. Browse to select the SAS Real-Time Decision Manager Design Server EAR file from a location. Select **Next**.



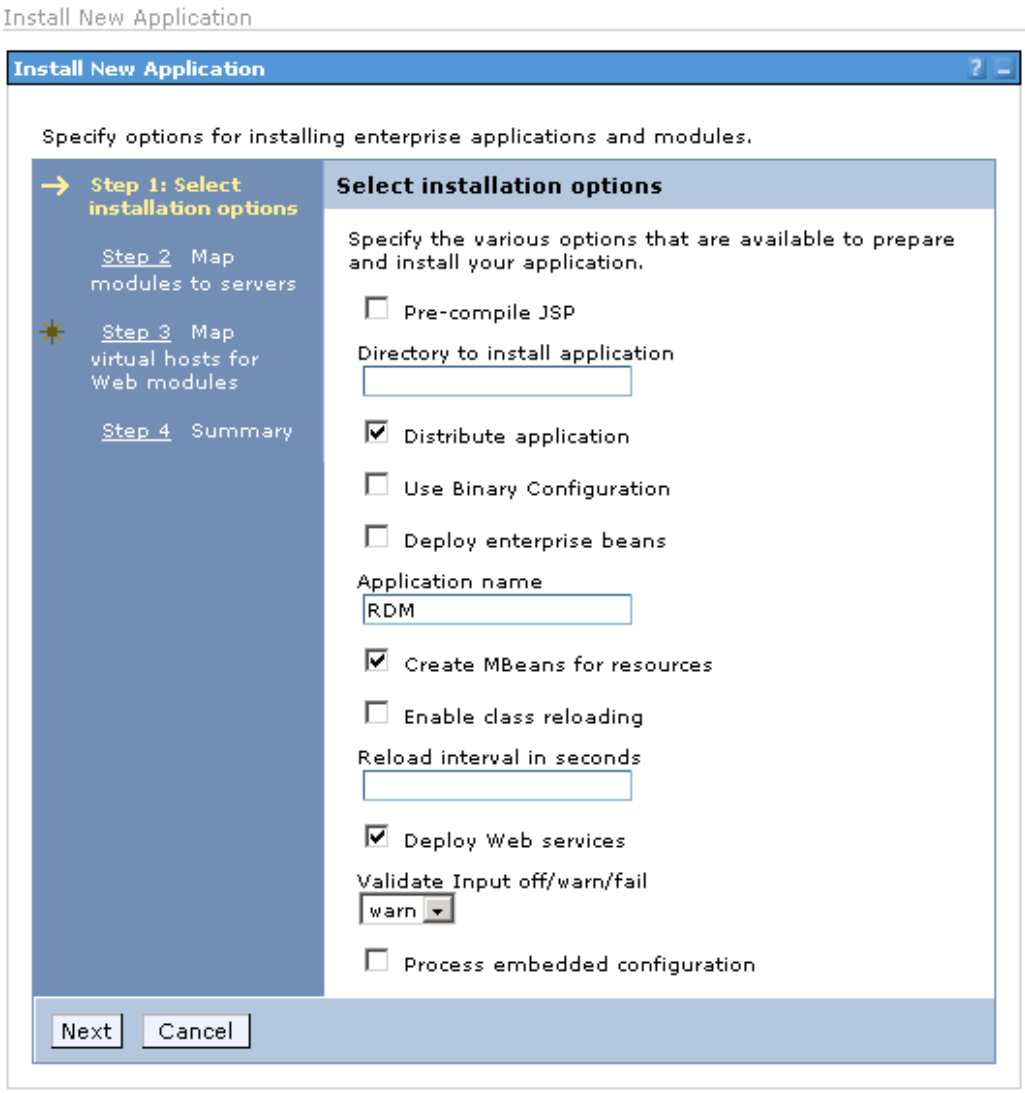

3. Select all of the defaults in this dialog box and select **Next**.



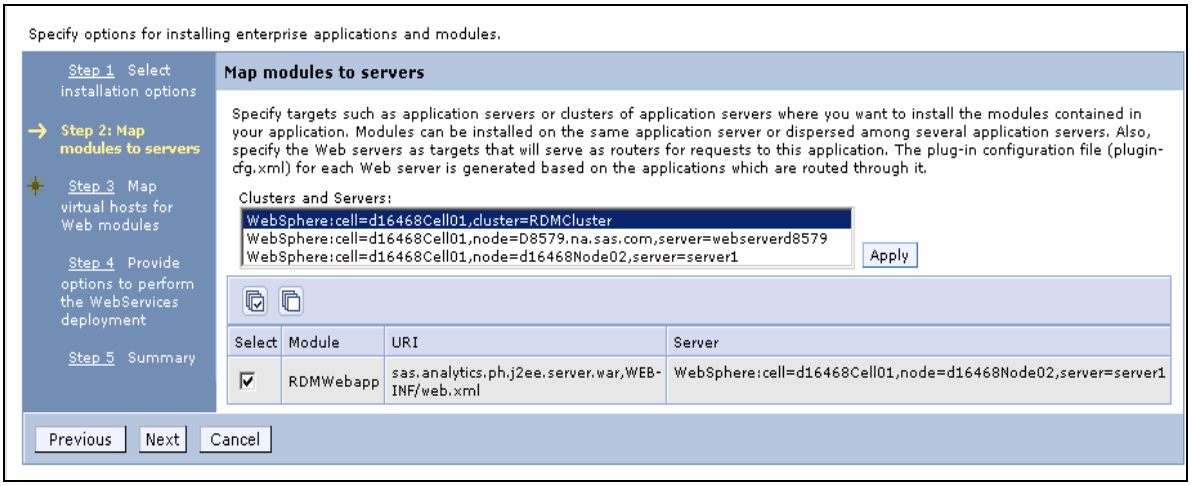

If your `WAS.policy` file has all permissions set, then the following dialog box appears:



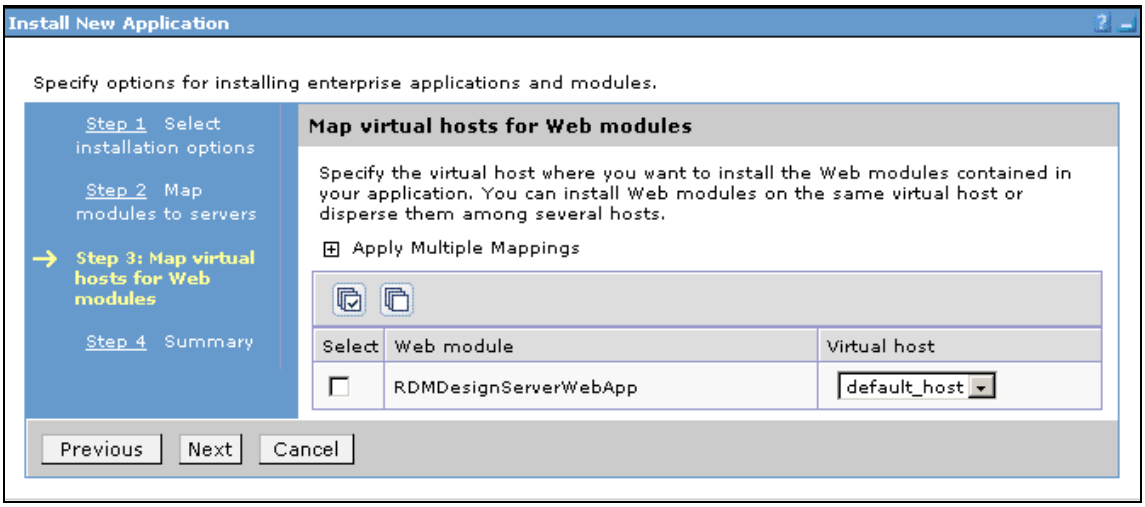

4. Select **Continue.**
5. In the "Select installation options" dialog box, select the checkbox **Deploy Web services**. The **Distribute application** and **Create MBeans for resources** checkboxes are checked by default. Select **Next**.



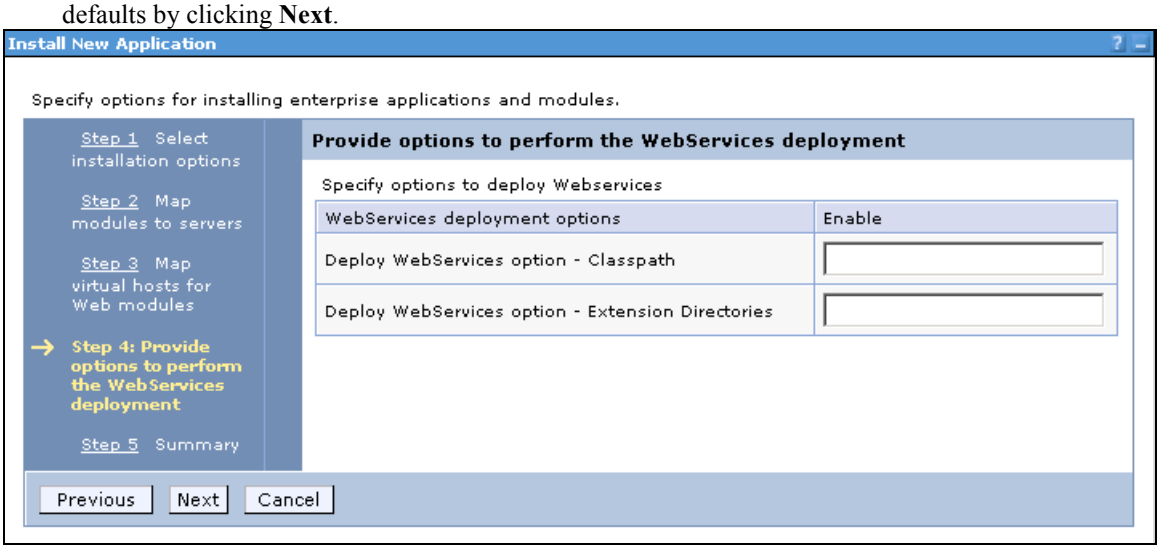

6. In the "Map modules to servers" dialog box, check the box beside **RDMDesignWebApp**. Click **Next.**



7. In the "Map virtual host for Web modules" dialog box, check the **Select** check box, ensure that **default_host** is visible in the drop down list, and click **Next.**




8. In the "Provide options to perform the Web Services deployment," dialog box, accept the defaults by clicking **Next**.




9. Review the Summary and select **Finish**.

10. Select **Save to Master Configuration**.

11. Select **Save**. Click **OK** when the messages finish displaying.

12. Verify that there were no errors by reviewing the `systemout.log` for your WebSphere Application Server profile. In a standard installation, the profile folder is located under *<WebSphere install root>*`/AppServer/profiles`.

## Start the SAS Real-Time Decision Manager Design Server

1. To start the SAS Real-Time Decision Manager Design Server application, within the WebSphere Application Server Admin Console navigate to **Enterprise Applications → RTDMDesign**. Select the check box next to the application and click **Start** at the top of the dialog box. This may take a few minutes.

2. Verify that the application started successfully by reviewing the systemout.log for your WebSphere Application Server profile. In a standard installation, the profile folder is located under `<WebSphere install root>/AppServer/profiles`.

## Verify the SAS Real-Time Decision Manager Design Server Deployment

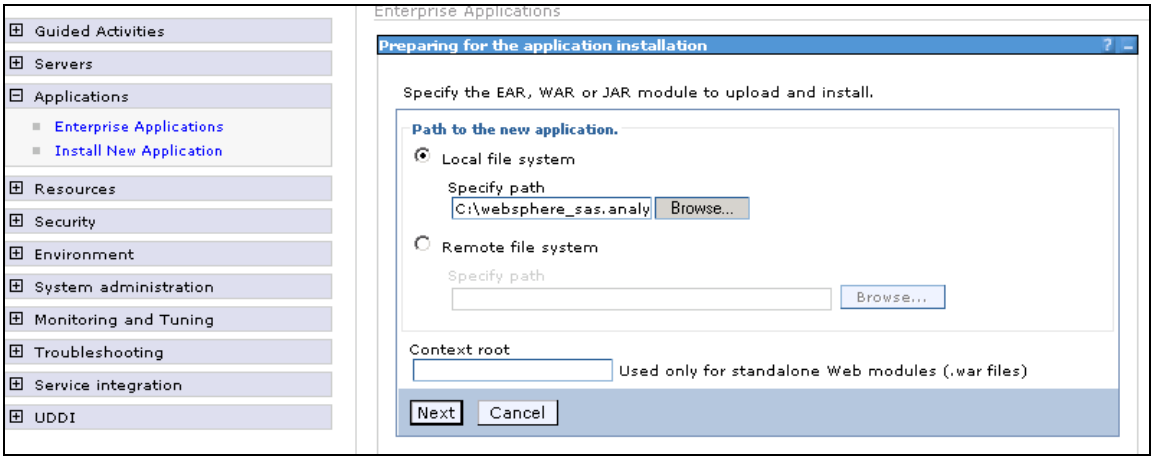
Enter **http://address:port/RTDMDesign/jsp/Diagnostics.jsp** into the address line of your Web browser. Address is the IP address of the machine on which the application server is installed. Port is the WC_defaulthost port (see the Record the Port Values and Administration Servlet URL section). This Web page displays information about the configuration of the application. It also connects to Remote services, OMR, and WebDAV, and returns the status of those connections.

**Note on integration:** Web service requests are submitted to http://address:port/RTDMDesign/Design. SAS Customer Intelligence 5.4 submits Web service requests to this address in order to create, modify, delete, and validate SAS Real-Time Decision Manager artifacts.

## Deploy the SAS Real-Time Decision Manager Engine Server EAR into the WebSphere Environment

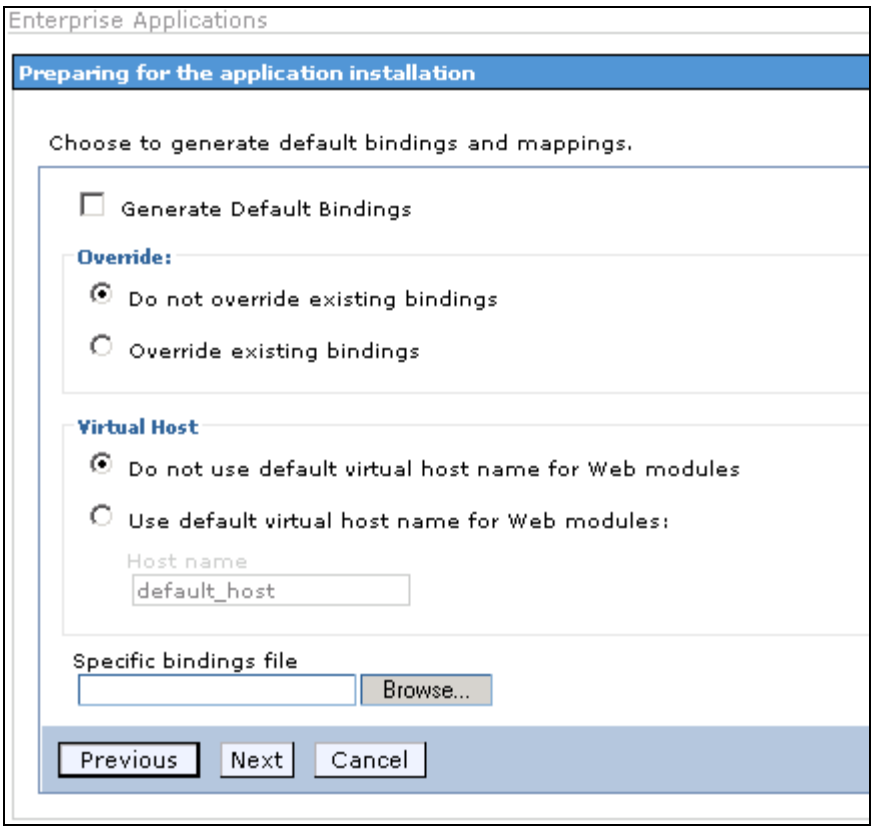
Start the WebSphere Application Server Admin Console.

1. Select **Install New Application**.
2. Browse to select the SAS Real-Time Decision Manager Engine Server EAR file from a location. Click **Next**.

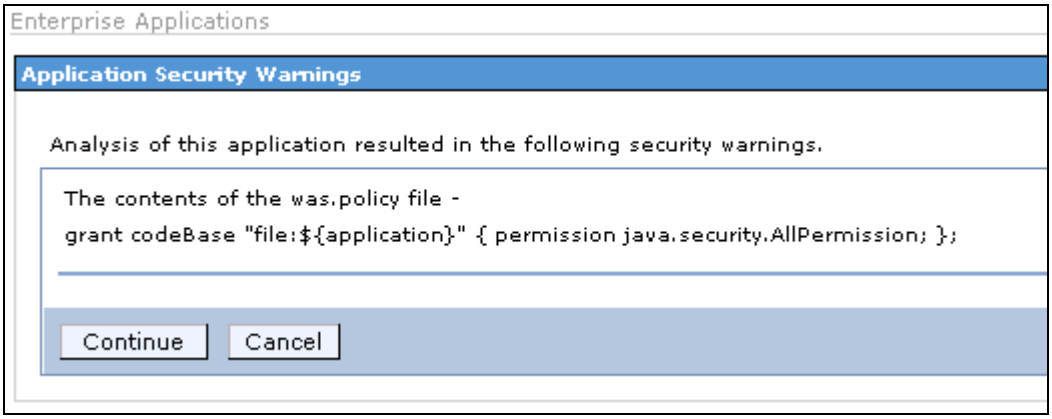


3. It is recommended to accept all of the defaults on the following dialog box

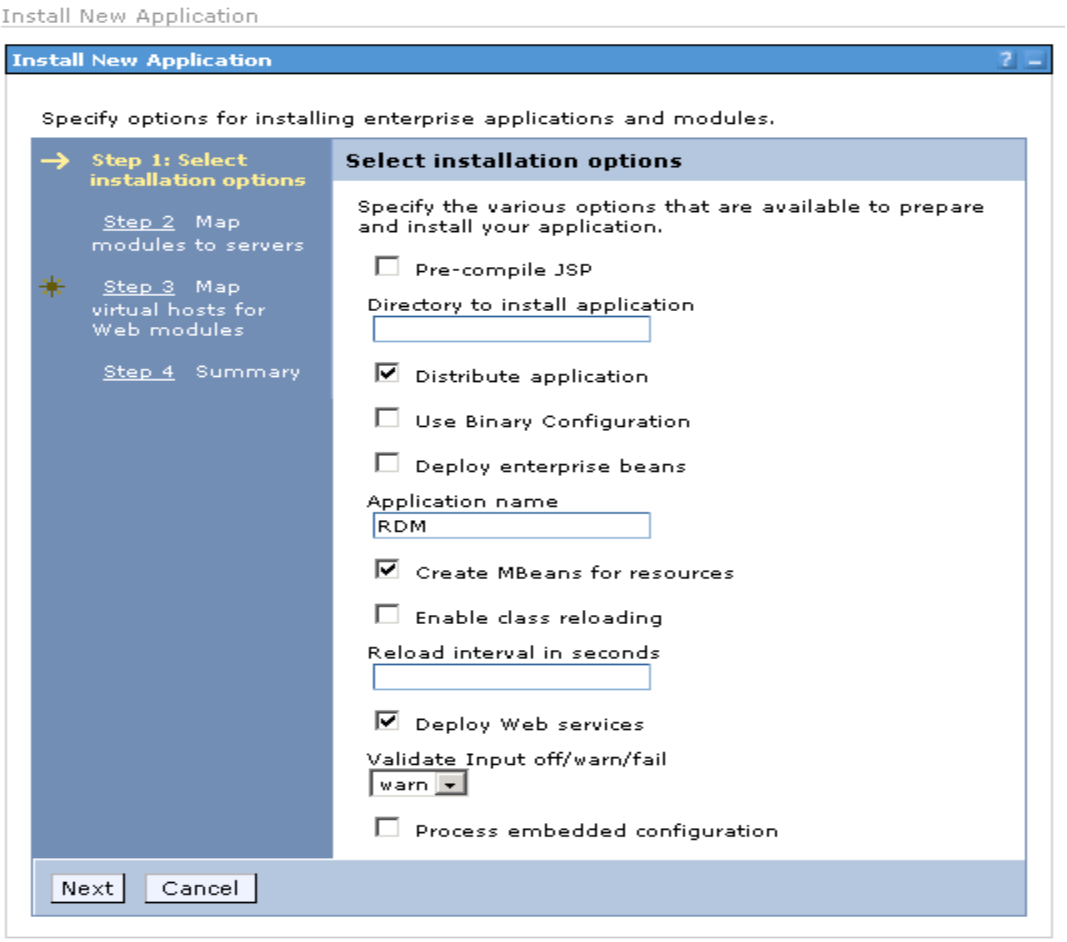
**Note:** Since you defined RDM_HOST in a previous step, you have the option to not use the default virtual host.



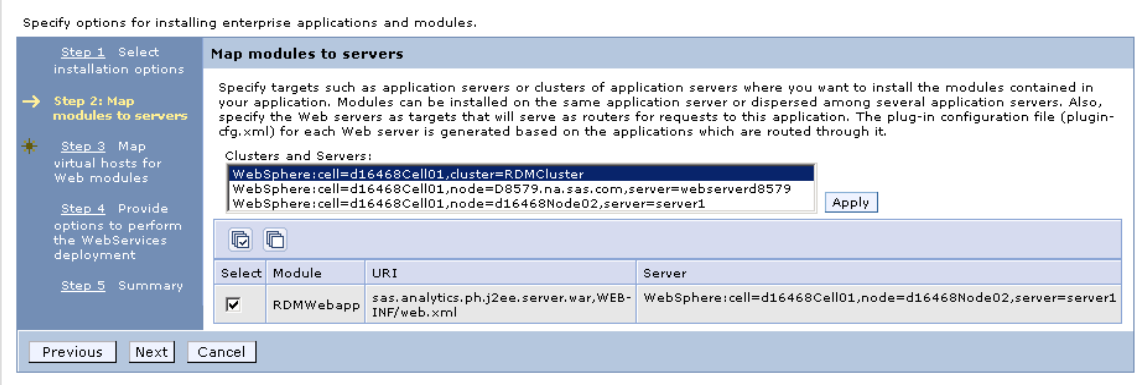

The following dialog box is displayed when the `WAS.policy` file has all permissions set.




4. Select **Continue**.
5. In the following **Select installation options** dialog box, check **Deploy Web Services**. The **Distribute application** and **Create MBeans for resources** options are checked by default. Select **Next**.



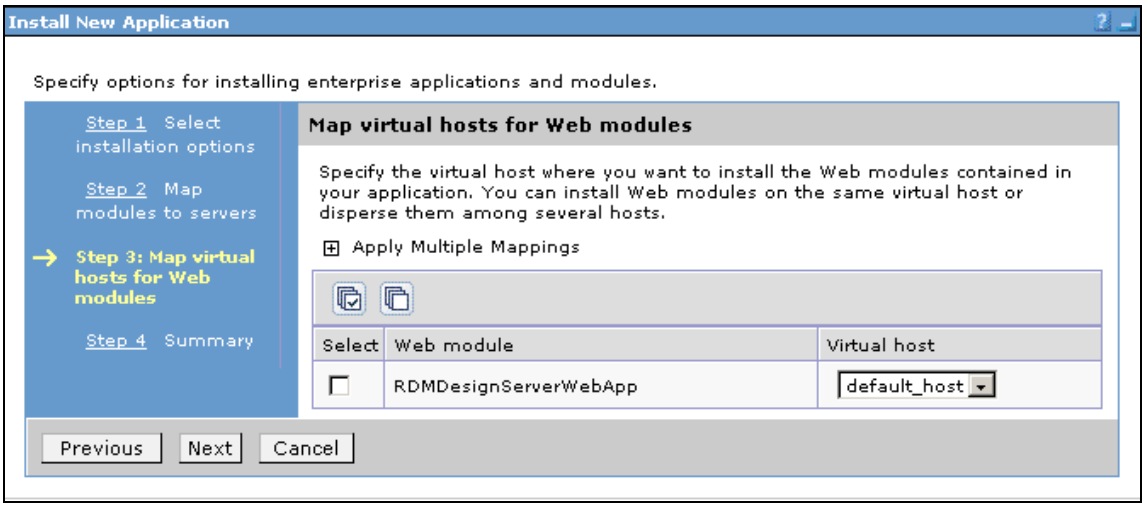

6. In the **Map modules to servers** dialog box, check **RDMWebApp** and click **Apply**.



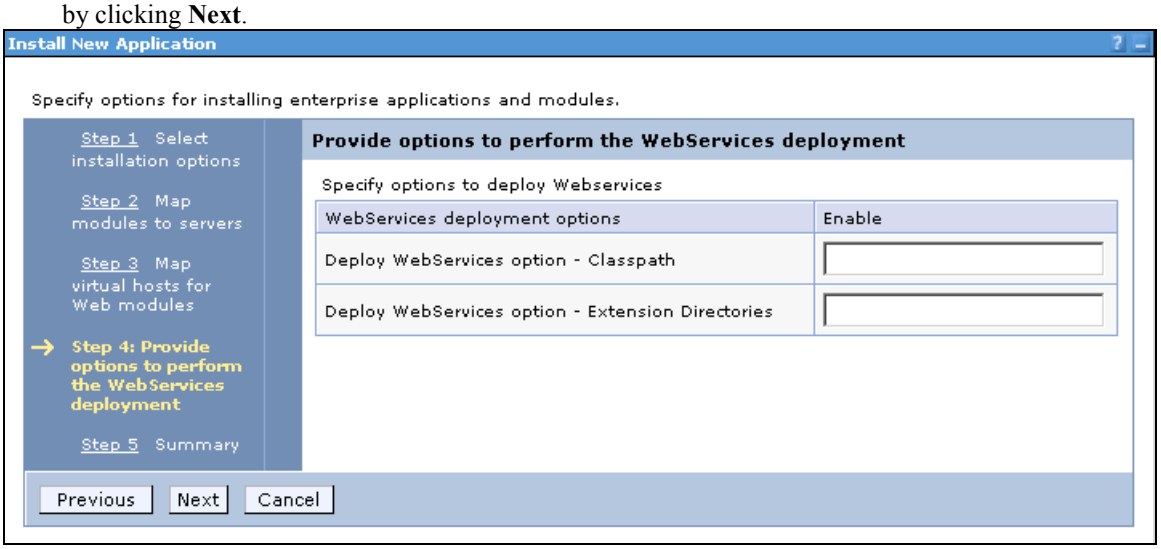

Make sure that the following value was changed.

Select **Next**.

7. In **Map virtual hosts for Web modules**, select **Next**.




8. In **Provide options to perform the Web Services deployment** dialog box, accept the defaults by clicking **Next**.



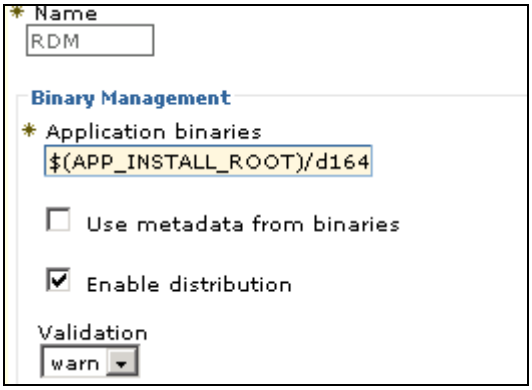

9. Review the summary and select **Finish**.

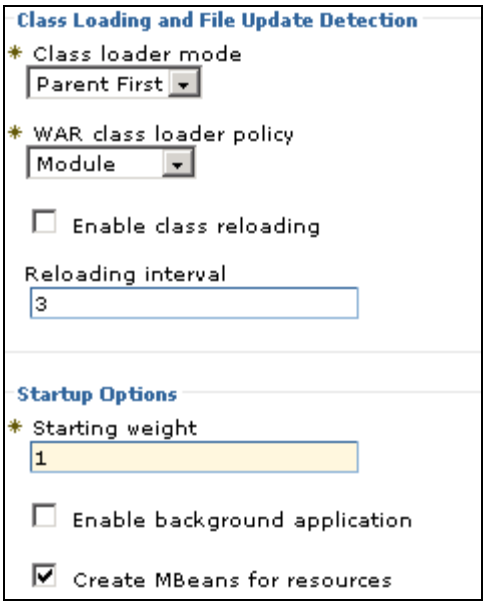
10. Select **Save to Master Configuration**.

11. Select **Save**. Click **OK** after the messages finish displaying.

12. Verify that there were no errors by reviewing the `systemout.log` for your WebSphere Application Server profile. In a standard installation, the profile folder is located under *<WebSphere install root>*`/AppServer/profiles`.

13. To view the general properties of the SAS Real-Time Decision Manager application that you just deployed: From the WebSphere Application Server Administration Console, select **Enterprise Applications** and click the RDM application.




**Note:** You should not have to change any of these values unless you are sure that your changes will better optimize the application.

## Start the SAS Real-Time Decision Manager Engine Server Cluster

The cluster must be started in order for the application to be started on all of the servers in the cluster.

1. Open WebSphere Admin Console.
2. Start the RDM Cluster by navigating to **Servers→Clusters**.
3. Select the check box of the cluster and click **Start** at the top of the dialog box. This may take a few minutes.

Verify that the application started successfully on each server in the cluster by reviewing the systemout.log for your WebSphere Application Server profile.

# Install and Configure a Production Environment

A typical installation consists of development, test, and production environments. The test environment is optional. Test and production environments are installed and configured similarly, so the following instructions apply to both environments. However, you may choose to allocate fewer hardware, network, or database resources to your test environment than to your production environment.

## Install the WebSphere Application Server

Install IBM WebSphere Application Server 6.1 with fix pack 21. For information on how to install WebSphere, refer to your IBM vendor. See also: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/tpro_instancesdmgr.html

## Create a Deployment Manager Profile

For production, the SAS Real-Time Decision Manager engine should be configured with middle tier and SAS server-tier clusters. A high-performance database configuration is also highly recommended.

Configure the IBM WebSphere Application Server with Network Deployment. Determine your cluster topology configuration. Refer to the topic, Planning to create application server environments, on the IBM information center Web site at: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/welcome_nd.html

These instructions provide a brief overview of how to define a WebSphere Deployment Manager profile. For details, see: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/tpro_instancesdmgr.html

1. Start your WebSphere Application Server if it is not already started.
2. Start the profile creation wizard.
3. Select **Next**.
4. Select **Create a deployment manager profile.**
5. Enter a profile name: _____
6. Select your profile directory: _____
7. Select your Node Name:_____
8. Select your Host Name:_____
9. Select **Next**.
10. Select **Next**.
11. Select **Next**.
12. Select **Finish**.
13. Run the installation verification to ensure that the server was created successfully.
14. Start the Server.

Determine how many application servers you will define in your deployment manager profile.

Following the instructions below, define as many application server profiles as you wish to include in your deployment manager profile. See also:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tpro_instancessaappserv.html

1. Start your WebSphere Application Server if it is not already started.
2. Start the profile creation wizard.
3. Select **Next**.
4. Select the radio button **Create an application server profile.**
5. Enter a profile name: Profile Name: _____
6. Select your profile directory:_____
7. Select your Node Name:_____
8. Select your Host Name:_____

After you have created the application server profiles, you will need to add the nodes into the cell that you defined when you created your deployment manager profile. To define a more complex network deployment environments, see Network Deployment at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/welcome_nd.html

## Install the IBM HTTP Server

Install the IBM HTTP Server by following the instructions under IBM HTTP Server for WebSphere Application Server Version 7.0 at

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/welcome_nd.html.

If the Web server is remote, refer to *Setting up a remote Web server*. If the Web server is on the same machine as the deployment manager, refer to *Setting up a local Web server*.

Make a note of the IBM HTTP server installation path to the log file, configuration file and the plug-ins properties file.

After you configure the WebSphere Application Server with Network Deployment, you should always select **Save to Master Configuration** and select **Synchronize with Nodes** after each configuration change in the administration console.

## Record the Port Values and the Administration Servlet URL

If you followed the previous instructions to install WebSphere Application Server and create a Deployment Manager profile, use the last scenario below. If you are working with a previously installed WebSphere Application Server, find out from your WebSphere administrator which of the following four WebSphere Application Server configurations exists at your site:

- Application Server
- Application Server and Deployment Manager
- Clustered Application Server and Deployment Manager
- Clustered Application Server and Deployment Manager with HTTP Server

Follow the instructions for your particular WebSphere Application Server configuration.

## Application Server

To obtain the port numbers required by the SAS Real-Time Decision Manager install:

1. Open the WebSphere Administrative Console.
2. Select **Servers.**
3. Select **Application Servers.**
4. Select your application server's name.
5. Select **Ports** under **Communications.**

Record the values for the following ports. They will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 2809 | |
| SOAP_CONNECTOR_ADDRESS | 8880 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9100 | |

Substitute the appropriate values (in bold) into the following URL and record it. It will be needed during the SAS Real-Time Decision Manager installation process.

http://application-server-name:WC-defaulthost-port/RTDM/Event

## Application Server and Deployment Manager

To obtain the port numbers required by the SAS Real-Time Decision Manager install:

1. Open the WebSphere Administrative Console.
2. Select **System Administration.**
3. Select **Deployment Manager.**
4. Select **Ports.**

Record the values for the following ports. They will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 9809 | |
| SOAP_CONNECTOR_ADDRESS | 8879 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9101 | |

Substitute the appropriate values (in bold) into the following URL and record it. It will be needed during the SAS Real-Time Decision Manager installation process.

http://application-server-name:WC-defaulthost-port/RTDM/Event

## Clustered Application Server and Deployment Manager

To obtain the port numbers required by the SAS Real-Time Decision Manager install:

1. Open the WebSphere Administrative Console.
2. Select **System Administration.**
3. Select **Deployment Manager.**
4. Select **Ports.**

Record the values for the following ports. They will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 9809 | |
| SOAP_CONNECTOR_ADDRESS | 8879 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9101 | |

Substitute the appropriate values (in bold) into the following URL and record it. The URL will be needed during the SAS Real-Time Decision Manager installation process.

http://application-server-name:WC-defaulthost-port/RTDM/Event

## Clustered Application Server and Deployment Manager with HTTP Server

To obtain the port numbers required by the SAS Real-Time Decision Manager install:

1. Open the WebSphere Administrative Console.
2. Select **System Administration.**
3. Select **Deployment Manager.**
4. Select **Ports.**

Record the values for the following ports. The values will be needed during the SAS Real-Time Decision Manager installation process.

| Port Name | Default Port | Actual Port |
|---|---|---|
| BOOTSTRAP_ADDRESS | 9809 | |
| SOAP_CONNECTOR_ADDRESS | 8879 | |
| WC_defaulthost | 9080 | |
| ORB_LISTENER_ADDRESS | 9101 | |

Substitute the appropriate values (in bold) into the following URL and record it. It will be needed during the SAS Real-Time Decision Manager installation process.

http://HTTP-server-name/RTDM/Event

## Create a New Cluster

Below are brief instructions for defining a cluster. These instructions assume that you have already added your nodes when you defined your deployment manager profile. For more information on creating your cluster, see http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/trun_wlm_cluster.html

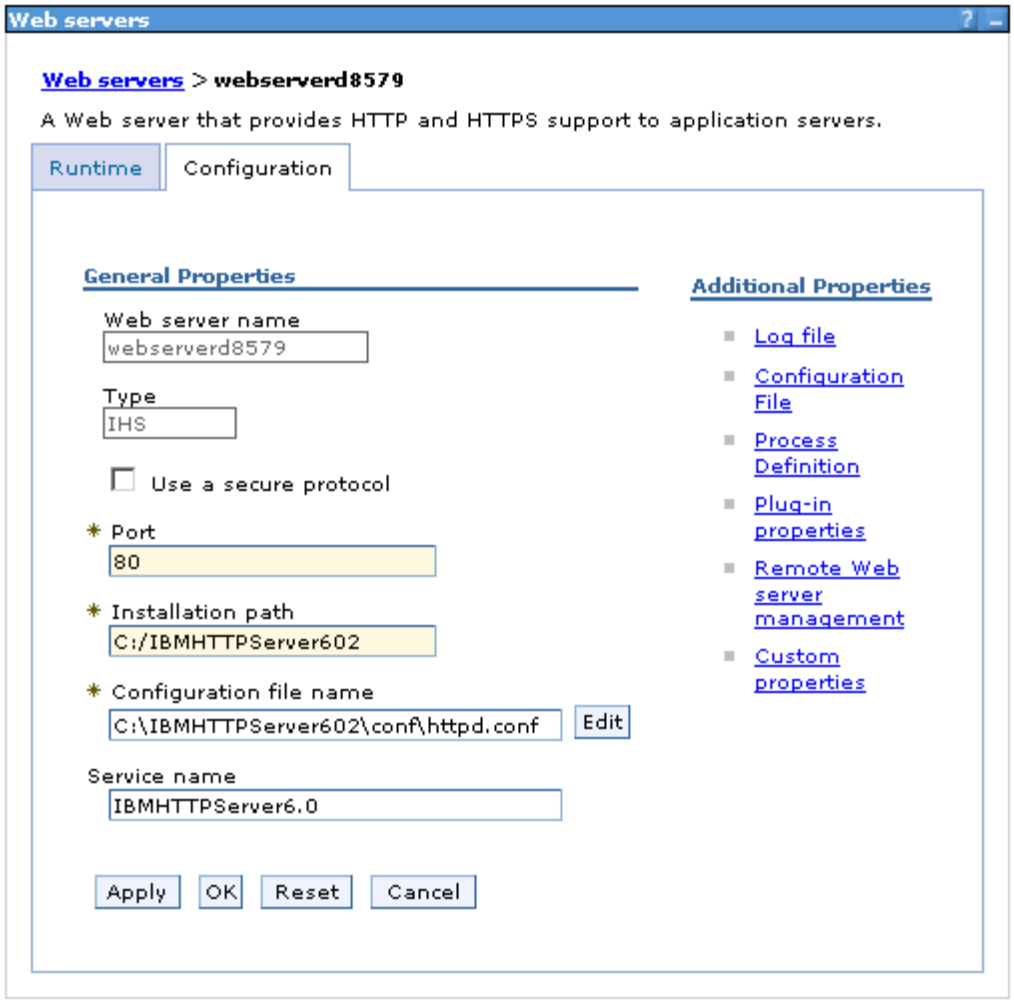
1. Open the WebSphere Administration Console.
2. In the navigation tree, select **Servers → Clusters.**
3. Select **New.**
4. Enter basic cluster information.
    a. Cluster Name:_____
    b. Check **Create a replication domain for this cluster.**
5. Select **Next**.
6. Repeat the following for each member to add to the cluster.
    a. Enter a member name: _____
    b. Select the node: _____
    c. Select a weight to assign to the member: _____
    d. Select the check box: **Generate Unique HTTP Ports.**
    e. Click **Apply.**
7. Select **Next**.
8. Select **Finish**.
9. Select **Save** in the Message box or boxes.
10. Select the check box: **Synchronize changes with Nodes.**
11. Select **Do not include an existing server in this cluster.**

## Create a Web Server Definition

If the Web server is remote, then locate a file on your remote HTTP server called `configurewebserverYOURHOST.bat` and copy it to the `Appserver\bin` directory. Run this file to specify the Web server definition.

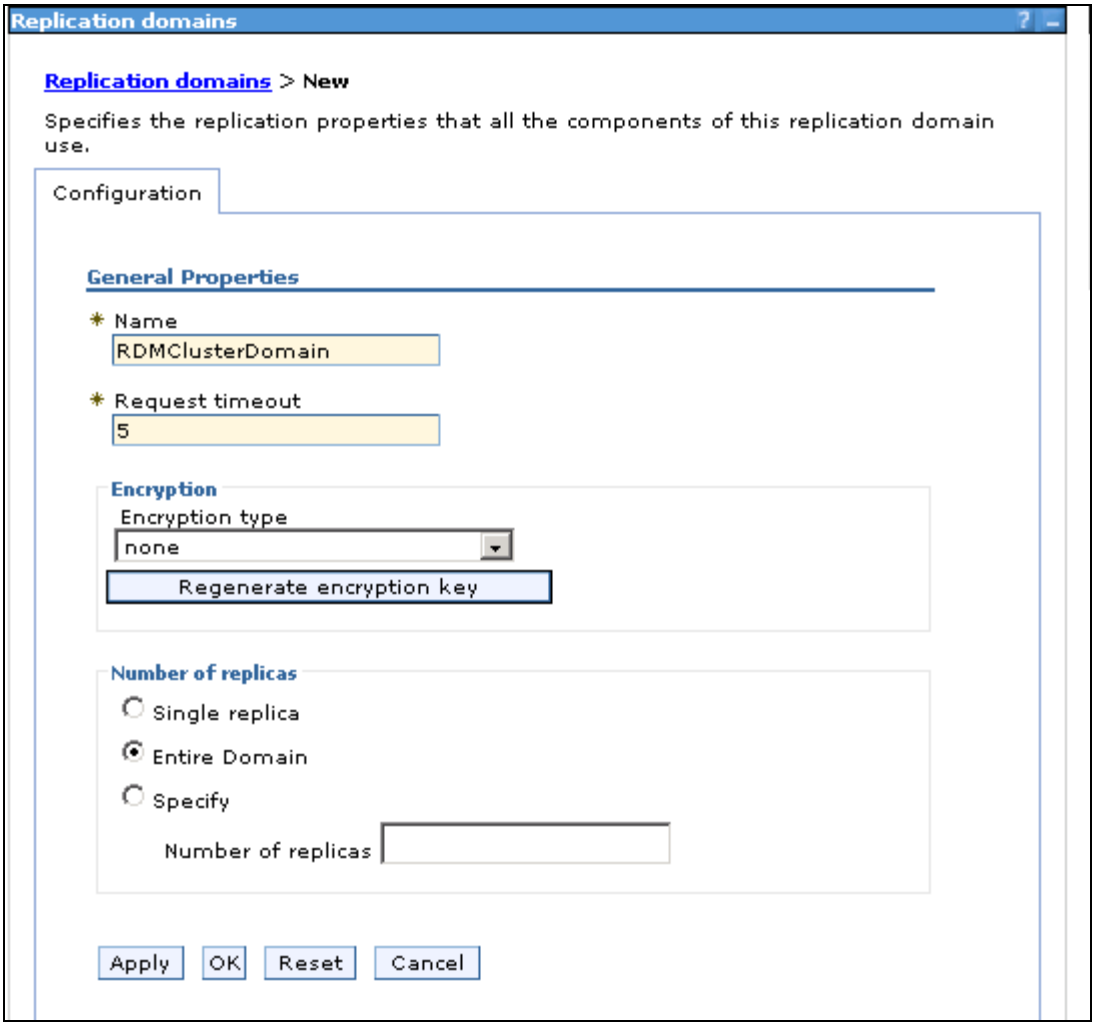
If the Web server is local, then you can create a new Web server definition from the WebSphere Administration Console. Navigate to **Servers → Web servers**. Enter the Web server name, port, installation path, and configuration file name.

Under **Additional Properties**, select the log file and configuration file and make sure that you can see the log file and the configuration file from here. Review the plug-in properties to make sure that the path information is correct.




Enter the options. Sample options are included above. Select **Apply** and **OK**.

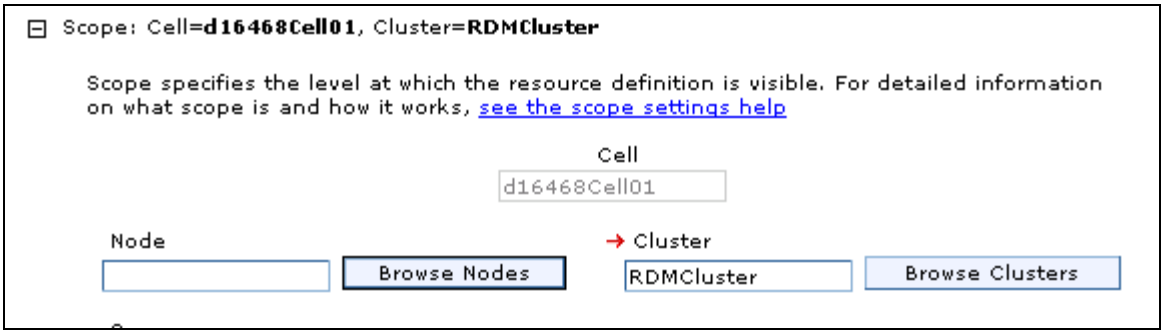
If you did not check the box, then create a replication domain when your cluster was created. Then, complete this step to configure the WebSphere Cache in the cluster. From the admin console, navigate to **Environment→Replication domains**. Click **New** and create a new replication domain with replicas in the entire domain. Select **Entire Domain**. Click **OK** to save and then save to the master configuration and synchronize the nodes.




## Define a New Object Cache Instance

These instructions provide a brief overview for defining an object cache instance. The object cache is used by the SAS Real-Time Decision Manager Engine Server to cache its data items.

1. Open the WebSphere Administation Console.
2. Navigate to **resources→cache instances→Object Cache Instances** and click **New**.
3. Set the scope of the object cache instance by selecting the cluster as the scope.
4. Click **Browse Cluster**, navigate to the cluster that was previously defined, and click **Apply**.



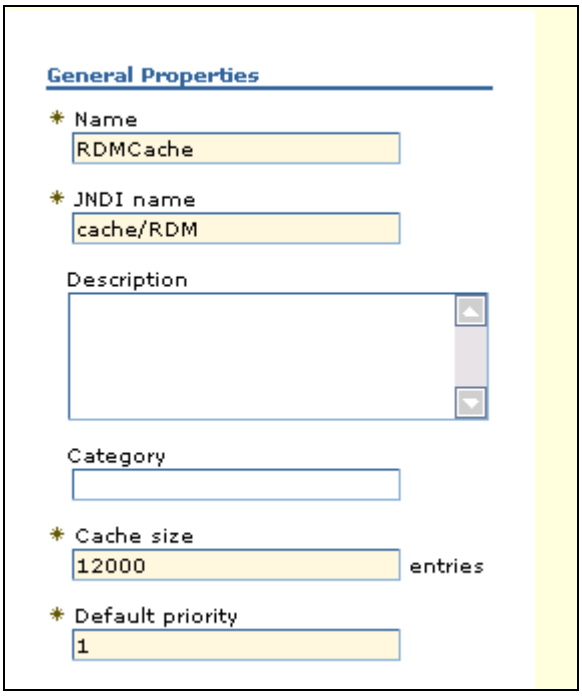

5. Using the sample options below, specify the cache instance name and the JNDI name. These options are used in the `cacheinstance.properties` file.

   Make a note of the cache name

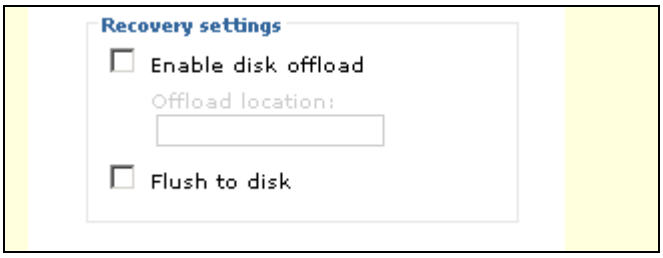
   _____

   and the JNDI name _____.

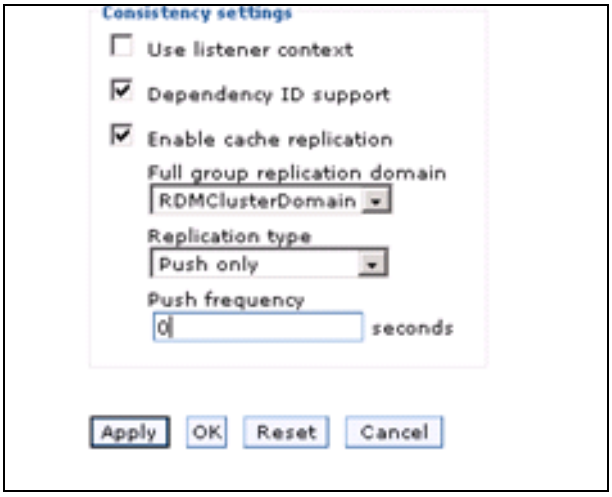
   These data will be used by the configuration utility that builds the EAR file.
6. Change the cache size to 12000.



The recovery settings should be set as follows:

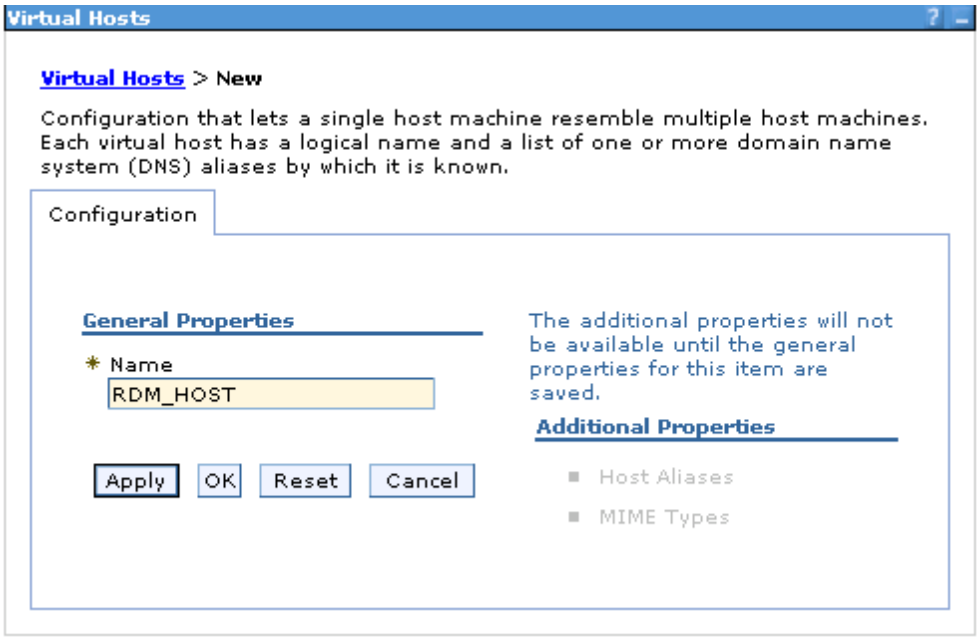



1. In the consistency settings, check the **Enable cache replication** box.
2. Use the replication domain that you defined above.
3. Set Replication type as **Push only.**
4. Set Push frequency to 0 (zero).




## Create and Configure a New Virtual Host

1. Open the Deployment Manager Administration Console.
2. Navigate to **Environment→Virtual Hosts→New Virtual Host.**
3. Enter `RDM_HOST` as the virtual host name.
4. Click **Apply** and click **Save configuration**.

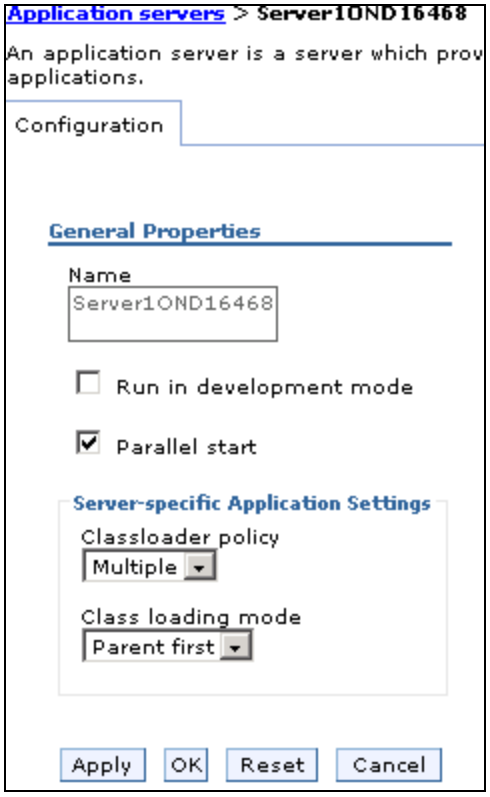



5. Click the newly created RDM_HOST and select **Host Aliases**. Enter the following (or similar) values, but be sure to record the values:

| | |
|---|---|
| RDM_HOST | *80 |
| | *9081 |
| | *9044 |

For each server that was added to the cluster above, change the Classloader policy to **Multiple** and the HTTP Transport port so that it matches the port number that has been defined in the virtual host RDM_HOST. For this example, we use ports 9081 and 9444 (for SSL).

6. Navigate to **Servers→Application servers.** Click each server and verify that the class loader policy is set to **Multiple**.




**Note:** Make sure that the **Application classloader policy** is set to **Multiple**. The setting for the "Application class loading mode", in the same panel (for the Application Server) doesn't matter, since it only applies when **application classloader policy** is set to Single.

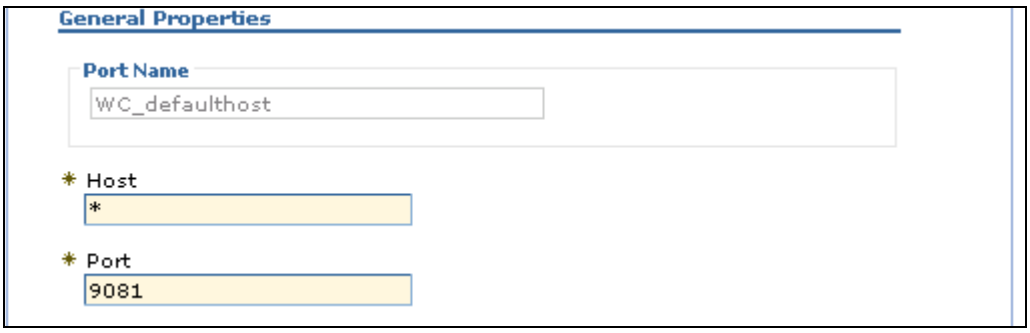
7. **Navigate to Servers→Application Servers.** Click on each server. For each server, under **Communications**, select **Ports**.


Verify the ports for the Web container. They should have the same ports as the virtual host.

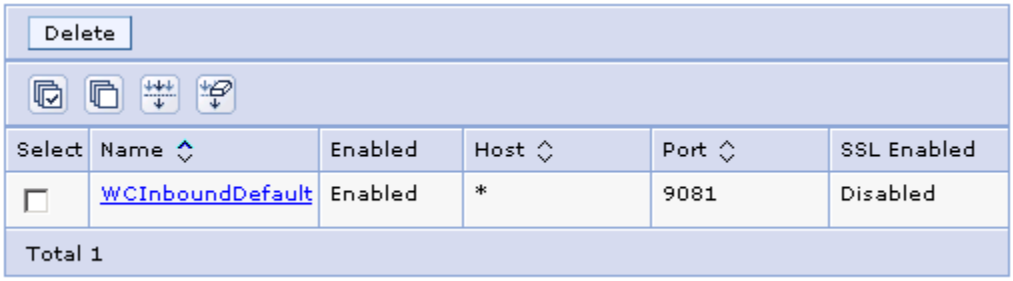
| | | | | |
|---|---|---|---|---|
| ☐ | WC_adminhost | * | 9062 | View associated transports |
| ☐ | WC_adminhost_secure | * | 9045 | View associated transports |
| ☐ | WC_defaulthost | * | 9081 | View associated transports |
| ☐ | WC_defaulthost_secure | * | 9444 | View associated transports |

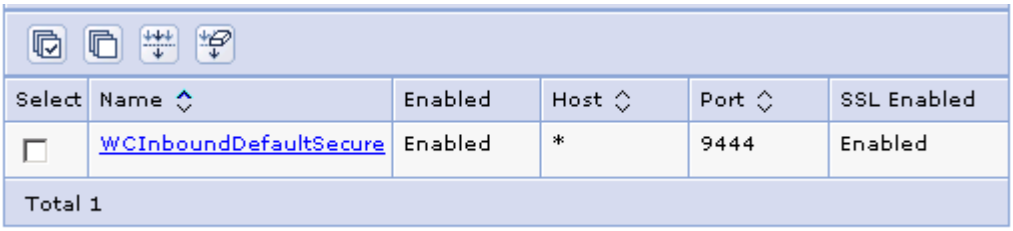
Total 15

To change the port numbers, select a port name (such as WC_defaulthost).




To view which settings are enabled, click **View associated transports** for WC_<*yourvirtualhost*>.




8. Open the folder <*WASROOT*>\appserver\bin. Run `backupconfig` to create a backup of the configuration.
9. Restart the Deployment Manager server.

## Install WebSphere MQ

Install WebSphere MQ 7.0.0.1 by following the instructions provided by IBM for installing on Solaris:
http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.amqdac.doc/sq10120_.htm

Separate MQ client and server installations are required.

WebSphere MQ can either be set up manually (see the following instructions), or automatically (using the SAS Deployment Wizard).

Follow these steps to manually set up your WebSphere MQ software queue manager,

queues, channel, and listener:

From a UNIX command line, follow these steps by issuing the following commands:

a. Create your SAS Real-Time Decision Manager Server Queue Manager:
```
crtmqm -q venus.queue.manager
```

b. Start your SAS Real-Time Decision Manager Server Queue Manager:
```
lstrmqm
```

c. Run the MQSC for Websphere MQ:
```
runmqsc
define qlocal (request.queue)
define qlocal ()
define listener (LISTENER1) trptype (tcp) control (QMGR) port
(1414)
start listener (LISTENER1)
define channel (CHANNEL1) chltype (svrconn) trptype (tcp)
start channel (CHANNEL1)
end
```

## Use the SAS Deployment Wizard to Install the SAS Real-Time Decision Manager Production Environment

### *Run the Installation Program*

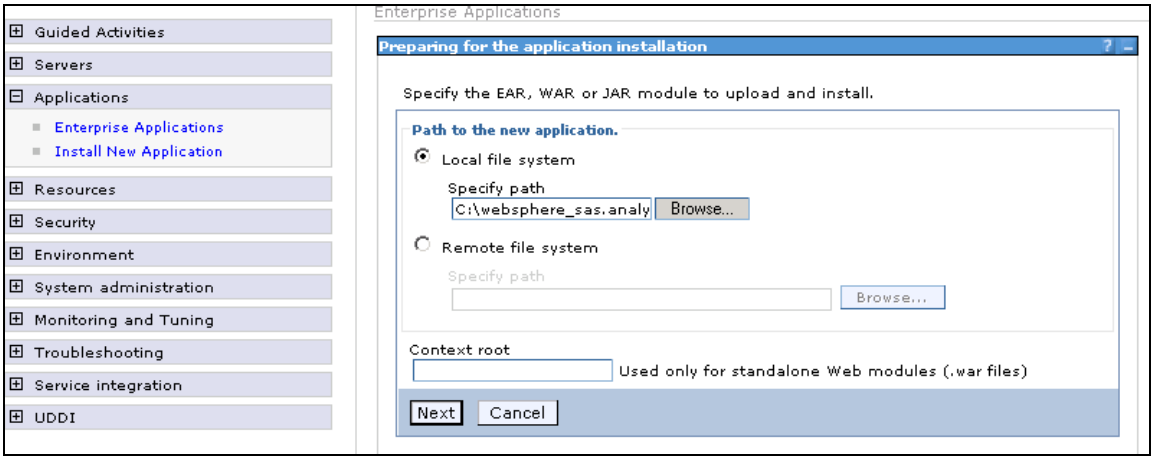
SAS support personnel will build a custom plan file for your specific SAS Real-Time Decision Manager installation. This planning file will be used as input to the SAS Deployment Wizard.

As the SAS Deployment Wizard executes, answer the prompts to provide the requested information about your installation environment.

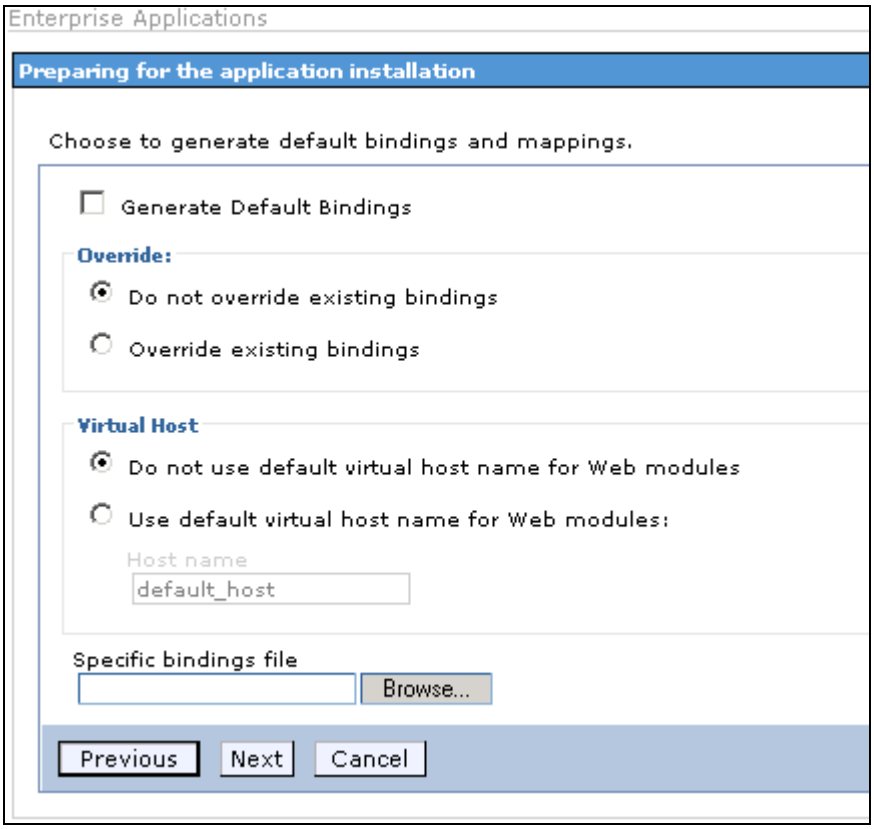
## Deploy the SAS Real-Time Decision Manager Engine Server EAR into the Clustered WebSphere Environment

Open the WebSphere Admin Console.

1. Select **Install New Application**.
2. Browse to select the SAS Real-Time Decision Manager Engine Server EAR file.
3. Click **Next**.

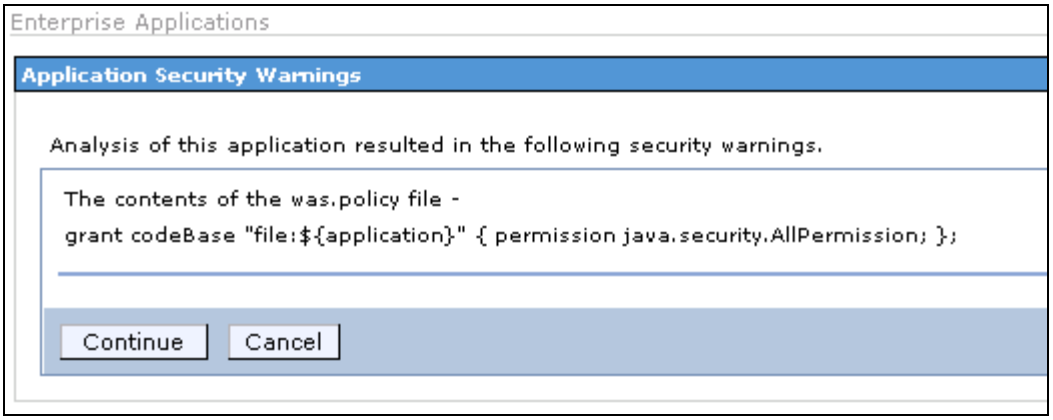



4. Select all of the defaults on this dialog box. If you want to use the default virtual host, then check **Use default virtual host name for Web modules.** If not then accept the default and select **Next**.

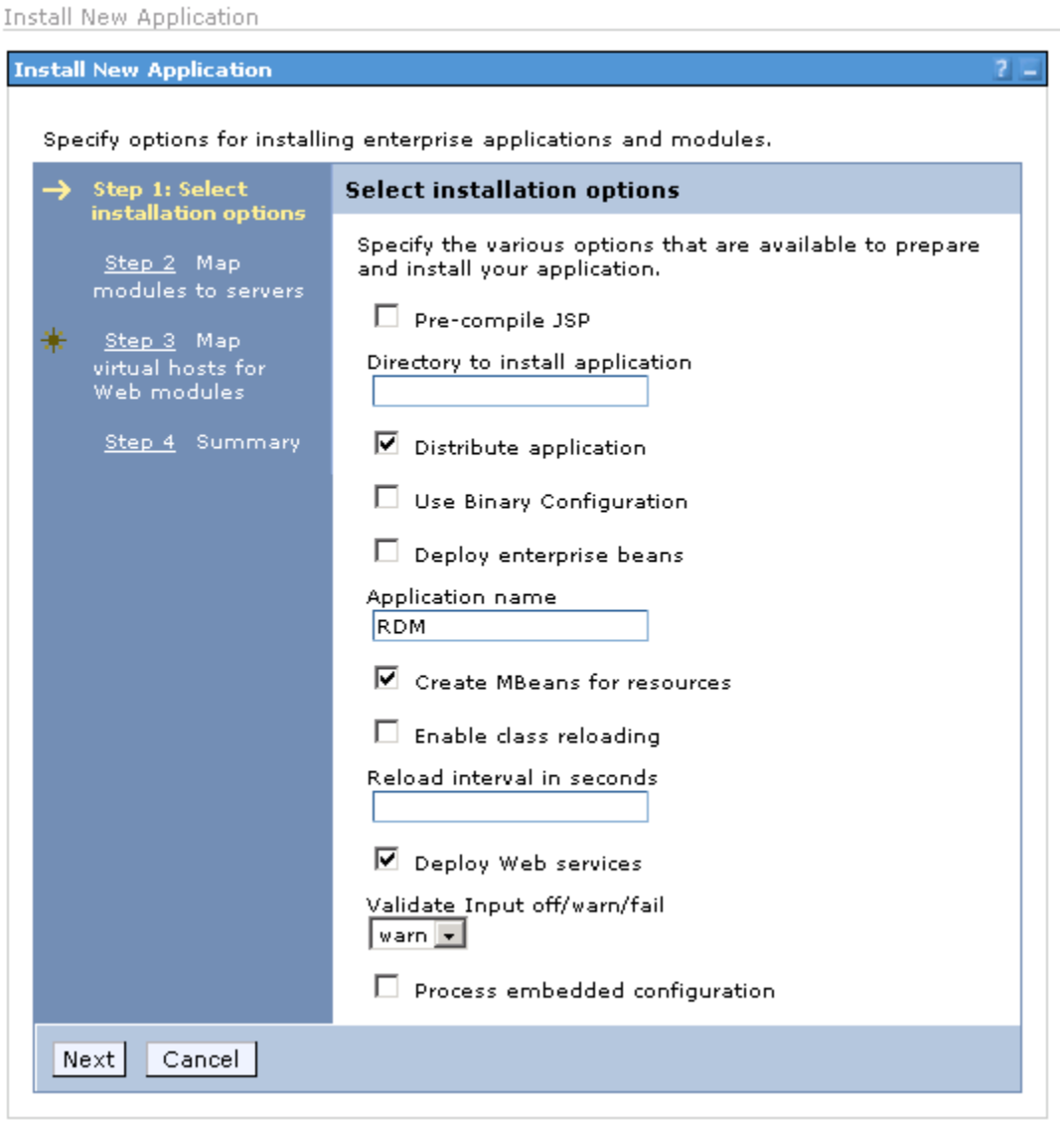



**Note:** Since we have defined a RDM_HOST in a previous section, do not use the default virtual host. Use the RDM_HOST instead.

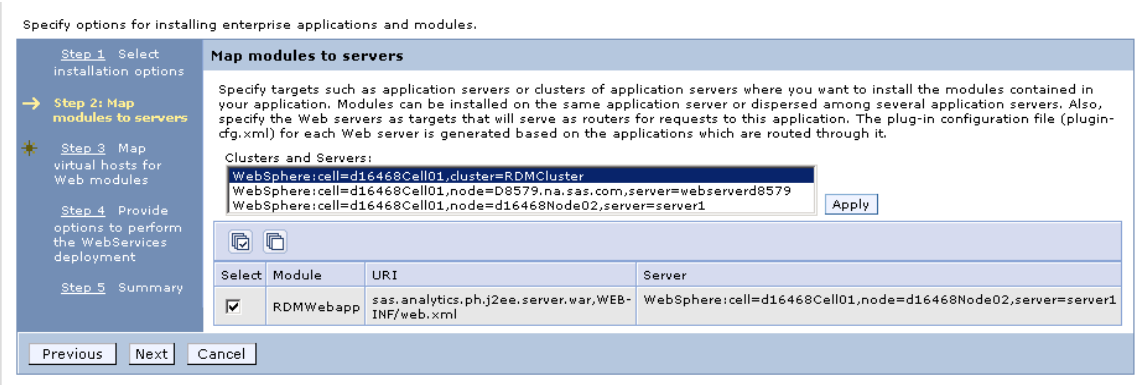
This dialog box below is displayed because the `WAS.policy` file has all permissions set.




5. Click **Continue**.
6. On the dialog box **Select installation options**, check the **Deploy Web services** check box.
7. Click **Next**.



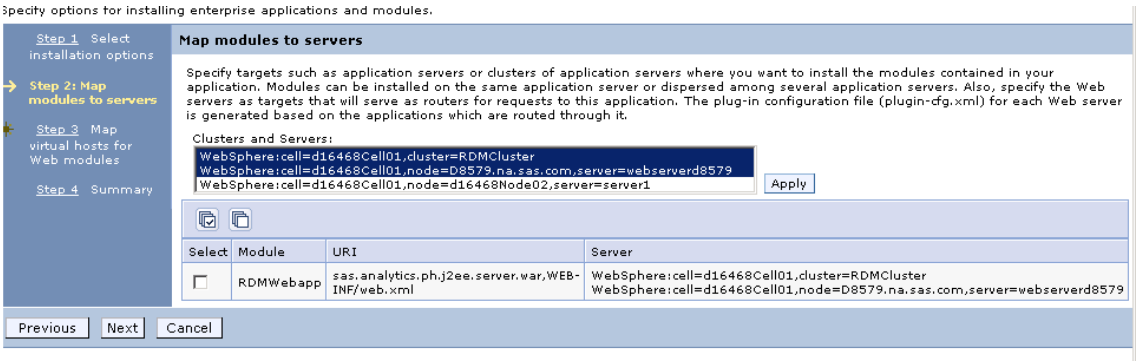

8. In the **Map modules to servers** dialog box, map the modules to the cluster name that you defined. The example cluster that was used in this document is RDMCluster. Select the RDMCluster node. Select the checkbox next to **RDMWebApp** and then click **Apply**.



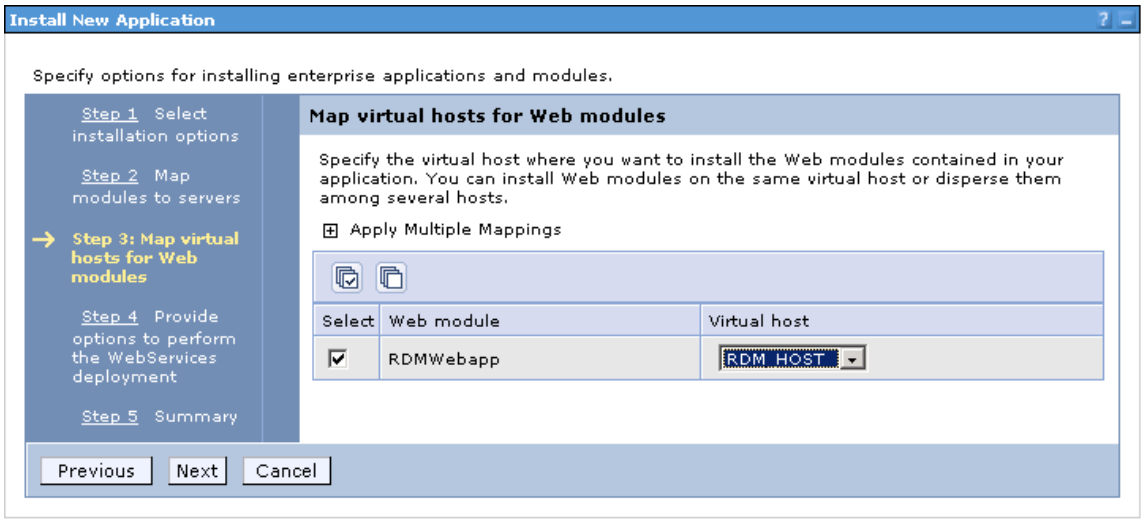
Verify that the value below was changed.

| Select | Module | URI | Server |
|---|---|---|---|
| ☐ | RDMWebapp | sas.analytics.ph.j2ee.server.war,WEB-INF/web.xml | WebSphere:cell=d16468Cell01,cluster=RDMCluster |

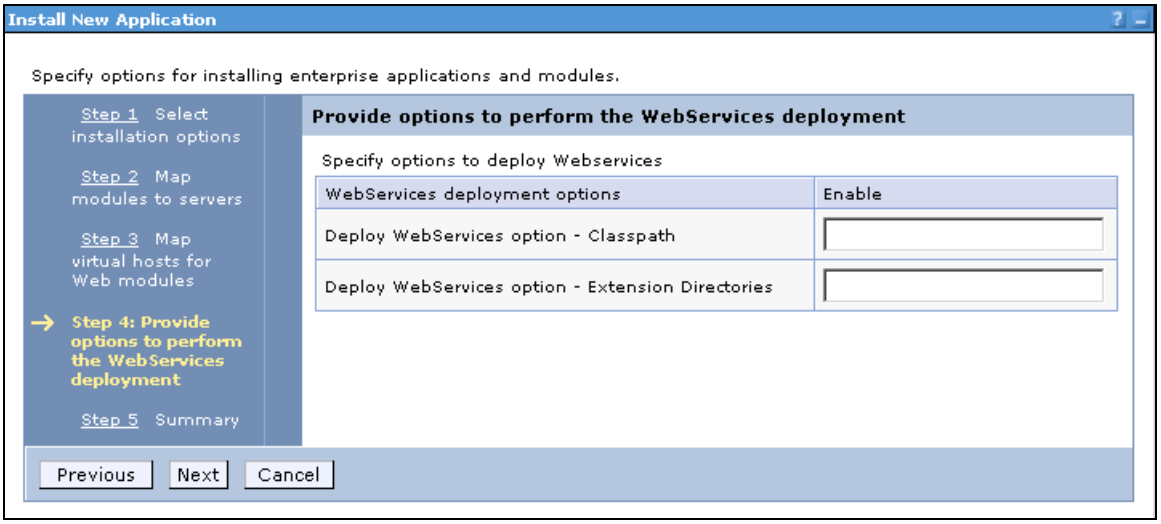
9. Map the application to the Web server module as well. Select the Web server and RDM Cluster. Click **Next**.




10. In the **Map virtual host for Web modules** step, select **RDM_HOST**. Click **Next**.




11. In **Apply options to perform the Web Services deployment**, accept the defaults by clicking **Next**.



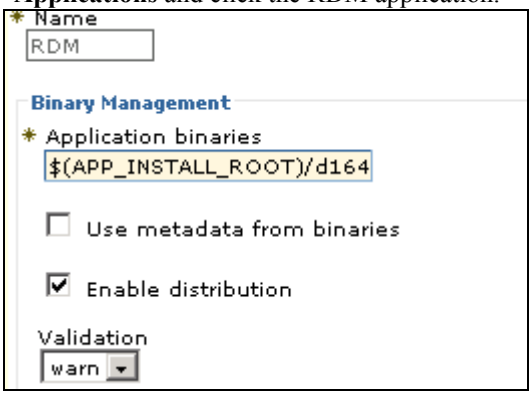

12. Review the summary and click **Finish**.

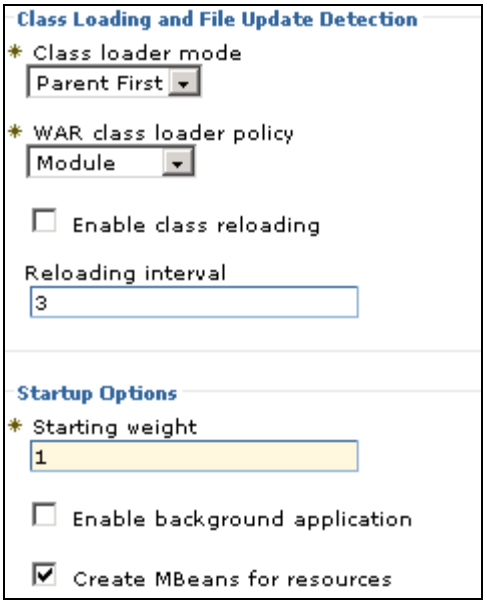
13. Click **Save to Master Configuration**.

14. Click **Save**. Click **OK** when the messages finish displaying.

15. Verify that there were no errors by reviewing the `systemout.log` for your WebSphere Application Server profile. In a standard installation, the profile folder is located under *`<WebSphere install root>`*`/AppServer/profiles`.

16. To view the general properties of the SAS Real-Time Decision Manager application that you just deployed: From the WebSphere Application Server Administration Console, select **Enterprise Applications** and click the RDM application.






**Note:** You should not have to change any of these values unless you are sure that your changes will better optimize the application.

## Create the HTTP Server Plug-in for Distributing Requests in the Cluster

1. Open WebSphere Administration Console.
2. Invoke **Servers→WebServers→GeneratePlugin** to create the HTTP Server plug-in for distributing requests in the cluster. This is done after the application is deployed so that the URLs that are exported by SAS Real-Time Decision Manager will be routed correctly.
   **GeneratePlugin** creates a file named `plugin-cfg.xml` at the following location:

   *<WebSphere home directory>*`/AppServer/profiles/`*<deployment manager profile>*`/config/cells/`*<cell name>*`/nodes/`*<name of the server containing WebSphere>*`/servers/`*<Web server definition>*

   **Note:** *The components in brackets should be substituted with the values appropriate for your environment.*

   Invoke **Servers→WebServers→PropagatePlugin** to propagate the HTTP Server plug-in. This is also done after the application is deployed so that the URLs that are exported by SAS Real-Time Decision Manager will be routed correctly.

3. Go to **System Administration→Nodes**, select all of the nodes where servers are located, and synchronize the changes.
4. Verify that the cluster has been started. If it has not started, then start the servers on each machine by selecting the server check boxes and clicking the **Start** button.

## Set Up the WebSphere MQ Server

If you have not already defined the Queue Manager, Queues, Listener, and Channels in WebSphere MQ, then you can run the SAS Deployment Wizard to do so.

When you are setting up the SAS Real-Time Decision Manager components that reside on the SAS server and the WebSphere MQ server on computers in different locales, you need to set the appropriate a coded character set ID (CCSID) for the SAS server.

The SAS server that contains the SAS Real-Time Decision Manager components is treated as a client of WebSphere MQ server. Therefore, it requires CCSID to be set to that of the server. This is done by defining the environment variable MQCCSID and setting it to the appropriate value. For details about setting the CCSID, see Chapter 11, "Using the message queue interface (MQI)" in the IBM document *WebSphere MQ Clients*. Also, the section, "Choosing client or server coded character set identifier (CCSID)" provides specific environment variable definitions.

For example, suppose that the WebSphere MQ server's CCSID is set to code page 850 and the SAS server that contains the SAS Real-Time Decision Manager components is installed under Windows. You can set the environment variable to 850 by setting the value in the environment variables section on the **Advanced** tab of the System properties dialog box.

**Note:** You must restart the SAS server Object Spawner for this to take effect.

If the SAS server that contains the SAS Real-Time Decision Manager components is installed under Solaris, then you can set the environment variable by typing the following on the command line before starting the SAS server Object Spawner:

```
export MQCCSID=850
```

SAS Real-Time Decision Manager 5.4 supports ports with four digits. Therefore, configure the WebSphere MQ Server with a four-digit port to ensure that connections are successful with Real-Time Decision Manager.

## Configure SAS Object Spawner to Use Oracle

If Oracle is being used, add the following statements to the ObjectSpawner.sh script:

### *Example*

```
LD_LIBRARY_PATH=/opt/mqm/lib64 ; export LD_LIBRARY_PATH
# Source the Oracle environment
. /data/Oracle/ora10g/ora_env
```

## Start the SAS Object Spawner

You must start the SAS Object Spawner and Remote Services before starting the SAS Real-Time Decision Manager engine server. Log on to the SAS server allocated for decision flow testing within the development environment.

From your UNIX prompt, enter:

```
ObjectSpawner status
```

The response will indicate whether SAS Object Spawner is already running or not.

If SAS Object Spawner is not running, enter:

```
SCServer start
```

### Start the SAS Real-Time Decision Manager Engine Server Cluster

The cluster must be started in order for the application to be started on all of the servers in the cluster.

1. Open WebSphere Administration Console.
2. Start the RDM Cluster by navigating to **Servers→Clusters.**
3. Select the check box of the cluster and click **Start** at the top of the dialog box. This may take a few minutes.

Verify that the application started successfully on each server in the cluster by reviewing the systemout.log for your WebSphere Application Server profile.

### Verify the SAS Real-Time Decision Manager Engine Server Deployment

Verify that the engine server is deployed correctly.

1. From a Web browser, enter:
   `http://`*`hostname`*`:WC_Defaulthost_port/RTDM/Event`

   *Hostname* is the URL of the server on which the SAS Real-Time Decision Manager engine is deployed.

2. Supply the WC_defaulthost port number at the prompt. See the section Record the Port Values and Administration Servlet URL for the WC_Defaulthost port number.

The following test Web page will cause the demo flow to be executed. Successful execution of the flow is a good indication that you have deployed each tier successfully.

http://hostname:WC_Defaulthost_port/RTDM/Event

## Post-Installation Reconfiguration

### Design Server Reconfiguration

You will need to unconfigure and then reconfigure the SAS Real-Time Decision Manager design server using the SAS Deployment Wizard.

### Engine Server Reconfiguration

You will need to unconfigure and then reconfigure the SAS Real-Time Decision Manager engine server using the SAS Deployment Wizard.

CHAPTER 10

# Migration



If you are performing an upgrade of SAS Real-Time Decision Manager 5.3, it is recommend that you export your user-defined SAS Real-Time Decision Manager 5.3 flows, events, resources, and activities manually before applying the upgrade. To export these resources, perform the following steps using SAS Real-Time Decision Manager 5.3.

1. Open SAS Management Console. Be sure to select the metadata profile of a user who has edit permissions in the design and production repositories.
2. Navigate to each user-defined repository folder in System\Applications\SAS Real-Time Decision Manager\SAS Real-Time Decision Manager 5.3.

3. Right-click on the repository folder level and select the export package menu item. Repeat this step for each folder repository.
4. Select **OK**.

During the upgrade process, your SAS Real-Time Decision Manager 5.3 flows, events, resources, and activities are automatically imported into your SAS Real-Time Decision Manager 5.4 repository.

No changes are needed in the artifacts that have been loaded in the OMR.

# Single DATA Step Server (SDS)

Changes have been made to the hash objects used for input and output. In Real-Time Decision Manager 5.3m there was a single key variable, sc_name, and two data variables, sc_string and sc_number. To properly handle arrays and tables, two additional keys have been added in Real-Time Decision Manager 5.4, sc_row_number and sc_column_name. The data variables have been simplified to a single value for Real-Time Decision Manager 5.4, sc_data_value.

The following is an example of version 5.3:

```
declare hash sc_input_values();
 sc_rc = sc_input_values.defineKey('sc_name');
 sc_rc = sc_input_values.defineData('sc_string', 'sc_number');
 sc_rc = sc_input_values.defineDone();
```

The following is an example of version 5.4:

```
 declare hash sc_input_values();
 sc_rc = sc_input_values.defineKey('sc_name', 'sc_row_number',
'sc_column_name');
 sc_rc = sc_input_values.defineData('sc_data_value');
 sc_rc = sc_input_values.defineDone();
```

## Input

### Scalars

The following examples illustrate the differences in scalars between Real-Time Decision Manager 5.3 and 5.4.

## Version 5.3

### Numeric Values

```
sc_input_values.find(key: 'inputInt');
inputInt=sc_number;
```

### Character Values

```
sc_input_values.find(key: 'inputString');
inputString=sc_string;
```

## Version 5.4

### Both Numeric and Character Values

```
sc_input_values.find(key: 'inputInt', key: ., key: '');
inputInt=sc_data_value;
sc_input_values.find(key: 'inputString', key: ., key: '');
inputString=sc_data_value;
```

## Arrays

The possibility for name conflicts existed for array values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. The examples below assume a string array named myArray with two values.

## 5.3 Version

### Number of Rows

```
sc_input_values.find(key: 'myArray');
row_count=sc_number;
```

### Data Access

```
sc_input_values.find(key: 'myArray1');
value_row_1=sc_string;
sc_input_values.find(key: 'myArray2');
value_row_2=sc_string;
```

### Type Metadata

```
/* Not available in 5.3 */
```

## 5.4 Version

### Number of Rows

```
sc_input_values.find(key: 'myArray', key: ,, key: 'num.rows');
row_count=sc_data_value;
```

**Data Access**

```
sc_input_values.find(key: 'myArray', key: 1, key: '');
value_row_1=sc_data_value;
sc_input_values.find(key: 'myArray', key: 2, key: '');
value_row_2=sc_data_value;
```

**Type Metadata**

```
sc_input_values.find(key: 'myArray', key: ,, key: 'array.type');
type=sc_data_value;
```

## Tables

The possibility for name conflicts existed for table values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. The examples below assume a table named myTable with two rows and two columns, myInt of type Int and myString of type String.

## Version 5.3

In Real-Time Decision Manager 5.3, tables were set up as several arrays. The main table array held the number of columns and the column names. Each column was then represented as its own array.

**Number of Columns**

```
sc_input_values.find(key: 'myTable');
column_count=sc_number;
```

**Column Names**

```
sc_input_values.find(key: 'myTable1'); /* Name of column 1 */
column_name1=sc_string;
sc_input_values.find(key: 'myTable2'); /* Name of column 2 */
column_name2=sc_string;
```

**Column Types**

```
/* Not available in 5.3 */
```

**Number of Rows**

```
sc_input_values.find(key: 'myInt');
row_count=sc_number;
```

or:

```
    sc_input_values.find(key: 'myString');
    row_count=sc_number;
```

### Data Access

```
    sc_input_values.find(key: 'myInt1');
    myInt_row_1=sc_number;
    sc_input_values.find(key: 'myInt2');
    myInt_row_2=sc_number;
    sc_input_values.find(key: 'myString1');
    myString_row_1=sc_string;
    sc_input_values.find(key: 'myString2');
    myString_row_2=sc_string;
```

## Version 5.4

In Real-Time Decision Manager 5.4, tables are able to take advantage of the additional keys to remove all potential name conflicts.

### Number of Columns

```
    sc_input_values.find(key: 'myTable', key: ., key:
'num.columns');
    column_count=sc_data_value;
```

### Column Names

```
    sc_input_values.find(key: 'myTable', key: 1, key: ''); /* Name
of column 1 */
    column_name1=sc_data_value;
    sc_input_values.find(key: 'myTable', key: 2, key: ''); /* Name
of column 2 */
    column_name2=sc_data_value;
```

### Column Types

```
    sc_input_values.find(key: 'myTable', key: ., key: 'myInt'); /*
Type for column myInt */
    myInt_type=sc_data_value; /* Value of 'Int' */
    sc_input_values.find(key: 'myTable', key: ., key: 'myString');
/* Type for column myString */
    myString_type=sc_data_value; /* Value of 'String' */
```

#### Number of Rows

```
sc_input_values.find(key: 'myTable', key: ., key: 'num.rows');
row_count=sc_data_value;
```

#### Data Access

```
sc_input_values.find(key: 'myTable', key: 1, key: 'myInt');
myInt_row_1=sc_data_value;
sc_input_values.find(key: 'myTable', key: 2, key: 'myInt');
myInt_row_2=sc_data_value;
sc_input_values.find(key: 'myTable', key: 1, key: 'myString');
myString_row_1=sc_data_value;
sc_input_values.find(key: 'myTable', key: 2, key: 'myString');
myString_row_2=sc_data_value;
```

## Output

### Scalars

In addition to the change to the output hash object, the value that should be returned for a DateTime variable has also changed. In Real-Time Decision Manager 5.3, this was inconsistent in that the DateTime input variables were actual numeric SAS DateTime values. However, DateTime output variables were strings with the datetime19.0 format applied to them. This has been fixed in version 5.4 so that DateTime variables are numeric SAS DateTime values for both input and output.

The following examples illustrate the differences in scalars between Real-Time Decision Manager 5.3 and 5.4.

### Version 5.3

#### Numeric Values

```
sc_output_values.add(key: 'outputInt', data: '', data:
outputInt);
```

#### Character Values

```
sc_output_values.add(key: 'outputString', data: outputString,
data: .);
```

#### DateTime Values

```
sc_string=put(outputDateTime, datetime19.0);
sc_output_values.add(key: 'outputDateTime', data: sc_string,
data: .);
```

## Version 5.4

### Numeric Values

```
    sc_data_value=outputInt; /* Automatic Conversion */

    sc_output_values.add(key: 'outputInt', key: ., key: '', data:
sc_data_value);
```

### Character Values

```
    sc_output_values.add(key: 'outputString', key: ., key: '', data:
outputString);
```

### DateTime Values

```
    sc_data_value=outputDateTime; /* Automatic Conversion */

    sc_output_values.add(key: 'outputDateTime', key: ., key: '',
data: sc_data_value);
```

## Arrays

The possibility for name conflicts existed for array values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. The examples below assume a string array named myArray with two values.

## Version 5.3

### Number of Rows

```
    sc_output_values.add(key: 'myArray', data: '', data: row_count);
```

### Array Type

```
    /* Not set used in 5.3 */
```

### Data Values

```
    sc_output_values.add(key: 'myArray1', data: value_row_1, data:
.);

    sc_output_values.add(key: 'myArray2', data: value_row_2, data:
.);
```

## Version 5.4

### Number of Rows

```
    sc_data_value=row_count; /* Automatic Conversion */

    sc_output_values.add(key: 'myArray', key: ., key: 'num.rows',
data: sc_data_value);
```

#### Array Type

```
    sc_output_values.add(key: 'myArray', key: ., key: 'array.type',
data: 'String');
```

#### Set Data Values

```
    sc_output_values.add(key: 'myArray', key: 1, key: '', data:
value_row_1);
    sc_output_values.add(key: 'myArray', key: 1, key: '', data:
value_row_2);
```

## Tables

The possibility for name conflicts existed for table values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. Additionally, in version 5.3 the type metadata for the table had to be set in the output table object in the middle tier. For version 5.4, this is no longer necessary; instead the type must be set in the activity code. This allows for greater flexibility in the output table. The examples below assume a table named myTable with two rows and two columns, myInt of type Int and myString of type String.

## Version 5.3

In version 5.3, tables were set as several arrays. The main table array held the number of columns and the column names. Each column was then represented as its own array.

#### Number of Columns

```
    sc_output_values.add(key: 'myTable', data: '', data:
column_count);
```

#### Column Names

```
    sc_output_values.add(key: 'myTable1', data: 'myInt', data: .);
    sc_output_values.add(key: 'myTable2', data: 'myString', data:
.);
```

#### Column Types

```
    /* Not set in 5.3 */
```

#### Number of Rows

```
    sc_output_values.add(key: 'myInt', data: '', data: row_count);
    sc_output_values.add(key: 'myString', data: '', data:
row_count);
```

### Data Values

```
    sc_output_values.add(key: 'myInt1', data: '', data:
myInt_row_1);

    sc_output_values.add(key: 'myInt2', data: '', data:
myInt_row_2);

    sc_output_values.add(key: 'myString1', data: myString_row_1,
data: .);

    sc_output_values.add(key: 'myString2', data: myString_row_2,
data: .);
```

## Version 5.4

In version 5.4, tables are able to take advantage of the additional keys to remove all potential name conflicts.

### Number of Columns

```
    sc_data_value=column_count; /* Automatic Conversion */

    sc_output_values.add(key: 'myTable', key: ., key: 'num.columns',
data: sc_data_value);
```

### Column Names

```
    sc_output_values.add(key: 'myTable', key: 1, key: '', data:
'myInt');

    sc_output_values.add(key: 'myTable', key: 2, key: '', data:
'myString');
```

### Column Types

```
    sc_output_values.add(key: 'myTable', key: ., key: 'myInt', data:
'Int');

    sc_output_values.add(key: 'myTable', key: ., key: 'myString',
data: 'String');
```

### Number of Rows

```
    sc_data_value=row_count; /* Automatic Conversion */

    sc_output_values.add(key: 'myTable', key: ., key: 'num.rows',
data: sc_data_value);
```

### Data Values

```
    sc_data_value=myInt_row_1; /* Automatic Conversion */
```

```
    sc_output_values.add(key: 'myTable', key: 1, key: 'myInt', data:
sc_data_value);
    sc_data_value=myInt_row_2; /* Automatic Conversion */
    sc_output_values.add(key: 'myTable', key: 2, key: 'myInt', data:
sc_data_value);
    sc_output_values.add(key: 'myTable', key: 1, key: 'myString',
data: myString_row_1);
    sc_output_values.add(key: 'myTable', key: 2, key: 'myString',
data: myString_row_2);
```

# Multiple DATA Step Server (MDS)

Changes have been made to the macro variables used for arrays and tables. In Real-Time Decision Manager 5.3 each value had its own macro variable. This allowed for the possibility of name conflicts. In version 5.4, array and table macro variables are encoded strings that can be decoded and re-encoded using the provided macro and link statements.

## Input

### Scalars

There are no changes to scalars in Real-Time Decision Manager 5.4.

### Arrays

The possibility for name conflicts existed for array values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. The examples below assume a string array named myArray with two values.

### Version 5.3

**Number of Rows**

```
    row_count=symget('myArray');
```

**Array Type**

```
    /* Not available in 5.3 */
```

**Data Access**

```
    value_row_1=symget('myArray1');
    value_row_2=symget('myArray2');
```

### 5.4 Version

**Number of Rows**

```
    data _null_;
        %sc_encode_decode
```

```
        sc_name='myArray';
        link sc_decode_array;

        sc_input_values.find(key: 'myArray', key: ., key:
'num.rows');
        row_count=sc_data_value;
    run;
```

**Array Type**

```
    data _null_;
        %sc_encode_decode

        sc_name='myArray';
        link sc_decode_array;

        sc_input_values.find(key: 'myArray', key: ., key:
'array.type');
        type=sc_data_value; /* Value of 'String' */
    run;
```

**Data Access**

```
    data _null_;
        %sc_encode_decode

        sc_name='myArray';
        link sc_decode_array;

        sc_input_values.find(key: 'myArray', key: 1, key: '');
        value_row_1=sc_data_value;
        sc_input_values.find(key: 'myArray', key: 2, key: '');
        value_row_2=sc_data_value;
    run;
```

## Tables

The possibility for name conflicts existed for table values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. The examples below assume a table named myTable with two rows and two columns, myInt of type Int and myString of type String.

## Version 5.3

In version 5.3, tables were setup as several arrays. The main table array held the number of columns and the column names. Each column was then represented as its own array.

**Number of Columns**

```
column_count=symget('myTable');
```

**Column Names**

```
column_name1=symget('myTable1');
column_name2=symget('myTable2');
```

**Column Types**

```
/* Not available in 5.3 */
```

**Number of Rows**

```
row_count=symget('myInt');
```

or:

```
row_count=symget('myString');
```

**Data Access**

```
myInt_row_1=symget('myInt1');
myInt_row_2=symget('myInt2');
myString_row_1=symget('myString1');
myString_row_2=symget('myString2');
```

## Version 5.4

**Number of Columns**

```
data _null_;
    %sc_encode_decode

    sc_name='myTable';
    link sc_decode_table;

    sc_input_values.find(key: 'myTable', key: ., key:
'num.columns');
    column_count=sc_data_value;
run;
```

**Column Names**

```
data _null_;
    %sc_encode_decode
```

```
        sc_name='myTable';
        link sc_decode_table;

        sc_input_values.find(key: 'myTable', key: 1, key: ''); /*
Name of column 1 */
        column_name1=sc_data_value;
        sc_input_values.find(key: 'myTable', key: 2, key: ''); /*
Name of column 2 */
        column_name2=sc_data_value;
    run;
```

### Column Types

```
    data _null_;
        %sc_encode_decode

        sc_name='myTable';
        link sc_decode_table;

        sc_input_values.find(key: 'myTable', key: ., key: 'myInt');
/* Type for column myInt */
        myInt_type=sc_data_value; /* Value of Int */
        sc_input_values.find(key: 'myTable', key: ., key:
'myString'); /* Type for column myString */
        myString_type=sc_data_value; /* Value of String */
    run;
```

### Number of Rows

```
    data _null_;
        %sc_encode_decode

        sc_name='myTable';
        link sc_decode_table;

        sc_input_values.find(key: 'myTable', key: ., key:
'num.rows');
        row_count=sc_data_value;
    run;
```

**Data Access**

```
    data _null_;
        %sc_encode_decode

        sc_name='myTable';
        link sc_decode_table;

        sc_input_values.find(key: 'myTable', key: 1, key: 'myInt');
        myInt_row_1=sc_data_value;
        sc_input_values.find(key: 'myTable', key: 2, key: 'myInt');
        myInt_row_2=sc_data_value;
        sc_input_values.find(key: 'myTable', key: 1, key:
'myString');
        myString_row_1=sc_data_value;
        sc_input_values.find(key: 'myTable', key: 2, key:
'myString');
        myString_row_2=sc_data_value;
     run;
```

# Output

## Scalars

In Real-Time Decision Manager 5.3, the DateTime input variables were actual numeric SAS DateTime values. However, DateTime output variables were strings with the datetime19.0 format applied to them. In version 5.4, DateTime variables are numeric SAS DateTime values for both input and output.

## Version 5.3

### All Types Excluding DateTime

```
call symputx('myInt', myInt);
```

### DateTime

```
call symputx('myDateTime', put(myDateTime, datetime19.0));
```

## Version 5.4

### All types

```
call symputx('myInt', myInt);
```

## Arrays

The possibility for name conflicts existed for array values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. The examples below assume a string array named myArray with two values.

## Version 5.3

### Number of Rows

```
call symputx('myArray', row_count);
```

### Array Type

```
/* Not used in 5.3 */
```

### Data Values

```
call symputx('myArray1', value_row_1);
call symputx('myArray2', value_row_2);
```

## Version 5.4

### Number of Rows

```
    data _null_;
        %sc_encode_decode

        row_count=sc_data_value; /* Automatic Conversion */
        sc_output_values.add(key: 'myArray', key: ., key:
'num.rows', data: sc_data_value);

        sc_name='myArray';
        link sc_encode_array;
    run;
```

### Array Type

```
    data _null_;
        %sc_encode_decode

        sc_output_values.add(key: 'myArray', key: ., key:
'array.type', data: 'String');

        sc_name='myArray';
        link sc_encode_array;
    run;
```

### Data Values

```
    data _null_;
        %sc_encode_decode

        sc_output_values.add(key: 'myArray', key: 1, key: '', data:
value_row_1);
        sc_output_values.add(key: 'myArray', key: 2, key: '', data:
value_row_2);

        sc_name='myArray';
        link sc_encode_array;
    run;
```

## Tables

The possibility for name conflicts existed for table values in Real-Time Decision Manager 5.3. This issue has been resolved in version 5.4. Additionally, in version 5.3, the type metadata for the table had to be set in the output table object in the middle tier. For version 5.4, this is no longer necessary. Instead, the type must be set in the activity code. This allows for greater flexibility in the output table. The following examples assume a table named myTable with two rows and two columns, myInt of type Int and myString of type String.

## 5.3 Version

In Real-Time Decision Manager 5.3 tables were set up as several arrays. The main table array held the number of columns and the column names. Each column was then represented as its own array.

**Number of Columns**

```
call symputx('myTable', column_count);
```

**Column Names**

```
call symputx('myTable1', 'myInt');
call symputx('myTable2', 'myString');
```

**Column Types**

```
/* Not used in 5.3 */
```

**Number of Rows**

```
call symputx('myInt', row_count);
call symputx('myString', row_count);
```

**Data Values**

```
call symputx('myInt1', myInt_row_1);
call symputx('myInt2', myInt_row_2);
call symputx('myString1', myString_row_1);
call symputx('myString2', myString_row_2);
```

## 5.4 Version

**Number of Columns**

```
data _null_;
    %sc_encode_decode

    sc_data_value=column_count; /* Automatic Conversion */
```

```
        sc_output_values.add(key: 'myTable', key: ., key:
'num.columns', sc_data_value);


        /* After table has been filled in */
        sc_name='myTable';
        link sc_encode_table;
    run;
```

### Column Names

```
    data _null_;
        %sc_encode_decode


        sc_output_values.add(key: 'myTable', key: 1, key: '',
'myInt');
        sc_output_values.add(key: 'myTable', key: 2, key: '',
'myString');


        /* After table has been filled in */
        sc_name='myTable';
        link sc_encode_table;
    run;
```

### Column Types

```
    data _null_;
        %sc_encode_decode


        sc_output_values.add(key: 'myTable', key: ., key: 'myInt',
'Int');
        sc_output_values.add(key: 'myTable', key: ., key:
'myString', 'String');


        /* After table has been filled in */
        sc_name='myTable';
        link sc_encode_table;
    run;
```

### Number of Rows

```
    data _null_;
        %sc_encode_decode

        sc_data_value=row_count; /* Automatic Conversion */
        sc_output_values.add(key: 'myTable', key: ., key:
'num.rows', sc_data_value);

        /* After table has been filled in */
        sc_name='myTable';
        link sc_encode_table;
    run;
```

### Data Values

```
    data _null_;
        %sc_encode_decode

        sc_data_value=myInt_row_1; /* Automatic Conversion */
        sc_output_values.add(key: 'myTable', key: 1, key: 'myInt',
sc_data_value);
        sc_data_value=myInt_row_2; /* Automatic Conversion */
        sc_output_values.add(key: 'myTable', key: 2, key: 'myInt',
sc_data_value);
        sc_output_values.add(key: 'myTable', key: 1, key:
'myString', myString_row_1);
        sc_output_values.add(key: 'myTable', key: 2, key:
'myString', myString_row_2);

        /* After table has been filled in */
        sc_name='myTable';
        link sc_encode_table;
     run;
```

CHAPTER 11

# Performance Tuning



## IBM WebSphere MQ Properties

Some properties cannot be changed when setting up the Queue Manager from the command line. In order to support higher performance, the following changes must be made manually:

Depending on the number of concurrent SAS Activity requests, some WebSphere MQ properties may need to be changed. If time-out exceptions are being received due to these activities, then some changes in the MQ properties may be required. These include the Max Handles, Max Channels, and Max Active Channels. By default, each is set to 100. If the Queue Manager was created using the SAS Real-Time Decision Manager Configuration tool, then the value of Max Handles will be set by default to 1500. See WebSphere MQ product documentation for information about how to tune for optimal performance in your specific environment. The values below may be used as a reasonable starting point.

You can change WebSphere MQ property values by following the instructions for the appropriate operating system:

**Windows**

1. Open the IBM WebSphere MQ Explorer program (Under Start > Programs > IBM WebSphere MQ).
2. Under the **Queue Managers** folder, right-click **QM_RTDM** and select **Properties** from the pop-up menu.
3. Select **Channels.** Set the **Max channels** to 1500, and the **Max active channels** to 1500. Select **TCP** and set the **TCP listener backlog** to 1500.

**UNIX**

1. Under /var/mqm/qmgrs/<Your QUEUE MANAGER NAME>, edit the qm.ini file.
2. Add the following values for channels:

```
MaxChannels=1500
MaxActiveChannels=1500
```

Add the following values for TCP:

```
ListenerBacklog=1500
```

### Example

```
   #*******************************************************************#
   #* Module Name: qm.ini                                             *#
   #* Type       : WebSphere MQ queue manager configuration file      *#
   #  Function   : Define the configuration of a single queue         *#
   #*              manager                                            *#
   #*******************************************************************#
   #* Notes      :                                                    *#
   #* 1) This file defines the configuration of the queue manager     *#
   #*                                                                 *#
   #*******************************************************************#
  ExitPath:
     ExitsDefaultPath=/var/mqm/exits/
     ExitsDefaultPath64=/var/mqm/exits64/
  #*
*#
  #*
*#
  CHANNELS:
     MaxChannels=1200
     MaxActiveChannels=1000
  #*
*#

  Log:
     LogPrimaryFiles=10
     LogSecondaryFiles=10
     LogFilePages=4096
     LogType=CIRCULAR
     LogBufferPages=4096
     LogPath=/var/mqm/log/QM_QMANAGER/
     LogWriteIntegrity=SingleWrite
  Service:
     Name=AuthorizationService
     EntryPoints=13
  ServiceComponent:
     Service=AuthorizationService
     Name=MQSeries.UNIX.auth.service
     Module=/opt/mqm/lib64/amqzfu
     ComponentDataSize=0
```

3. To update MaxHandles you need to run the following MQ runmqsc executable. This is an IBM executable that allows you to enter script commands. You should be able to do this from any directory as long as the IBM executables are in the path.

```
runmqsc [Queue Manager Name]
alter qmgr maxhands(1500)
end
```

# SAS Server Options

There are some options that can be edited to improve performance of the SAS Server. It is recommended that the logging be left on while setting up the server, and then turned off for production. When turned on, logging will significantly reduce overall system performance.

- Disable logging.
- The number of sessions is set to 1 initially in order to simplify the validation of the installation process. Increase the number of SAS sessions.
- The number of SAS sessions is related to the number of CPUs, and generally should be larger than the number of CPUs.
- The sasQSessionMax number represents the maximum number of concurrent SAS Sessions allowed on the SAS Server. Check the performance of the server while running at full speed. The CPU usage should be 80% or less.

The settings for logging and the settings for SAS sessions are located in the same Server Properties window in SAS Management Console. To change these settings, log in to SAS Management Console as the SAS Administrator and follow these steps:

1. On the **Plug-ins** tab, expand **Server Manager**.
2. Right-click MQ **Polling Server – SDS.**
3. Select **Properties**.
4. Select the **Options** tab
5. Edit the **Command** option by replacing:
   `-logcfgloc "D:\SAS\Config\Lev1\Applications\SASRealTimeServerConfig5.3\sdslogconfig.xml"`
   with:
   `-nolog`
6. Select **Advanced Options**.
7. Select the **Polling** tab.
8. Increase the **Maximum Sessions** value.
9. Select **OK**.
10. Select **OK**.

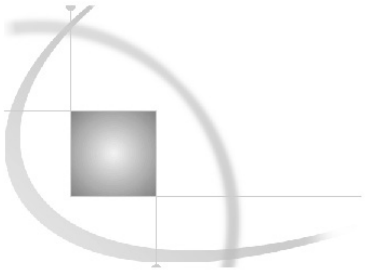
To change the logging and number of SAS Sessions for MQ Polling Server - rtdmtest1 – MDS, follow the previous steps.

Alternatively, the logging level can be changed on the SAS Server by editing the log configuration XML file. By default, the level is set to “info.” For production, the application logger should be changed to “Error.” This should be done for both the Single Data Step (SDS) and Multiple Data Step (MDS) log configuration files.

```
<!-- Application message logger -->
<logger name="App">
      <level value="Info"/>

  </logger>
```

After you increase your SAS sessions and disable your logging, then you must restart the SAS Object Spawner that was configured for SAS Real-Time Decision Manager. Restart the WebSphere MQ Server, queue manager and the object spawner so that it will be associated with the modified message queue polling server definitions.



# Glossary

**artifact**

an element of SAS metadata servers that might contain global variables, activities, events, system resources, or decision flow objects.

**campaign**

a planned set of one or more communications that are directed at a selected group of customers or potential customers for a commercial goal.

**contact history**

a record of the groups of individuals or organizations that have been identified to be contacted for a communication.

**counts metadata**

metadata that SAS Customer Intelligence Studio generates when some types of node diagrams are displayed. Counts metadata differs from the SAS metadata in the SAS Metadata Repository. Counts metadata typically consists of a list of values and a count of how often each value occurs in the database.

**custom property**

a user-defined property that you can create for information maps, relationships, data items, and filters, and for folders in an information map.

**data item**

in an information map, an item that represents either data (a table column, an OLAP hierarchy, or an OLAP measure) or a calculation. Data items are used for building queries. Data items are usually customized in order to present the data in a form that is relevant and meaningful to a business user.

**data set**

see *SAS data set*.

**database management system**

a software application that enables you to create and manipulate data that is stored in the form of databases.

**DBMS**

see *database management system*.

**format**

see *SAS format*.

**Java Naming and Directory Interface**

see *JNDI*.

**Java Virtual Machine**

see *JVM*.

**JNDI**

a standard extension to the Java platform that enables developers to create applications that can interact with a number of different naming services and directory services, such as the Domain Name System (DNS) and the Lightweight Directory Access Protocol (LDAP).

**JVM**

a program that interprets Java programming code so that the code can be executed by the operating system on a computer. The JVM can run on either the client or the server. The JVM is the main software component that makes Java programs portable across platforms. A JVM is included with JDKs and JREs from Oracle Corporation, as well as with most Web browsers.

**listen port**

in a network, a communications endpoint at which a server listens for requests for service from the client application. For example, a SAS Marketing Automation client application requests services from a BEA Weblogic Server Administration Server at the port number that has been designated as the listen port.

**log**

see *log file*.

**log file**

a file in which information about software processing is recorded as the processing occurs. A log file typically includes error messages and warning messages, but it can also include informational messages and statistics such as the number of records that have been processed or the amount of CPU time that a program required.

**macro variable**

a variable that is part of the SAS macro programming language. The value of a macro variable is a string that remains constant until you change it. Macro variables are sometimes referred to as symbolic variables.

**metadata**

descriptive data about data that is stored and managed in a database, in order to facilitate access to captured and archived data for further use.

**metadata server**

a server that stores information about servers, users, and stored processes and that provides this information to one or more client applications.

**middle tier**

in a SAS business intelligence system, the architectural layer in which Web applications and related services execute. The middle tier receives user requests, applies business logic and business rules, interacts with processing servers and data servers, and returns information to users.

**object spawner**

a program that instantiates object servers that are using an IOM bridge connection. The object spawner listens for incoming client requests for IOM services. When the spawner receives a request from a new client, it launches an instance of an IOM server to fulfill the request. Depending on which incoming TCP/IP port the request was made on, the spawner either invokes the administrator interface or processes a request for a UUID (Universal Unique Identifier).

**operating environment**

a computer, or a logical partition of a computer, and the resources (such as an operating system and other software and hardware) that are available to the computer or partition.

**plug-in**

a file that modifies, enhances, or extends the capabilities of an application program. The application program must be designed to accept plug-ins, and the plug-ins must meet design criteria specified by the developers of the application program. In SAS Management Console, a plug-in is a JAR file that is installed in the SAS Management Console directory to provide a specific administrative function. The plug-ins enable users to customize SAS Management Console to include only the functions that are needed.

**primary key**

a column or combination of columns that uniquely identifies a row in a table.

**response**

the reaction that an individual has to a campaign, such as requesting a quote, making an inquiry, opening an e-mail message, or buying the product.

**SAS data set**

a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

**SAS format**

a type of SAS language element that applies a pattern to or executes instructions for a data value to be displayed or written as output. Types of formats correspond to the data's type: numeric, character, date, time, or timestamp. The ability to create user-defined formats is also supported. Examples of SAS formats are BINARY and DATE.

**SAS Management Console**

a Java application that provides a single user interface for performing SAS administrative tasks.

**SAS Metadata Repository**

a file or collection of files that is used by the SAS Metadata server to store and retrieve metadata about application elements.

**schema**

a map or model of the overall data structure of a database. A schema consists of schema records that are organized in a hierarchical tree structure. Schema records contain schema items.

**spawner**

see *object spawner*.

# Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.

# SAS® Publishing Delivers!

SAS Publishing provides you with a wide range of resources to help you develop your SAS software expertise.
Visit us online at **support.sas.com/bookstore**.

## SAS® PRESS

SAS Press titles deliver expert advice from SAS® users worldwide. Written by experienced SAS professionals, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

**support.sas.com/saspress**

## SAS® DOCUMENTATION

We produce a full range of primary documentation:

- Online help built into the software
- Tutorials integrated into the product
- Reference documentation delivered in HTML and PDF formats—free on the Web
- Hard-copy books

**support.sas.com/documentation**

## SAS® PUBLISHING NEWS

Subscribe to SAS Publishing News to receive up-to-date information via e-mail about all new SAS titles, product news, special offers and promotions, and Web site features.

**support.sas.com/spn**

## SOCIAL MEDIA: JOIN THE CONVERSATION!

Connect with SAS Publishing through social media. Visit our Web site for links to our pages on Facebook, Twitter, and LinkedIn. Learn about our blogs, author podcasts, and RSS feeds, too.

**support.sas.com/socialmedia**

sas® THE POWER TO KNOW®