



THE
POWER
TO KNOW.

SAS[®] Decision Services 6.4

Administrator's Guide

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *SAS® Decision Services 6.4: Administrator's Guide*. Cary, NC: SAS Institute Inc.

SAS® Decision Services 6.4: Administrator's Guide

Copyright © 2015, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

August 2015

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Contents

<i>What's New in SAS Decision Services 6.4</i>	<i>vii</i>
<i>Accessibility</i>	<i>ix</i>
Chapter 1 • Overview of SAS Decision Services	1
What Is SAS Decision Services?	1
Decision Services Manager Plug-in for SAS Management Console	2
SAS Decision Services Repository	4
Chapter 2 • SAS Decision Services Concepts	7
Modes of Execution	8
Scalability and Failover	9
Deployment Topology	12
Example: The Decision Flow	16
Life Cycle of a Decision Flow	18
Decision Flows, Building Blocks, and Artifacts	20
Roles and Capabilities	27
SAS Decision Services Monitoring	28
Chapter 3 • Common Operations	33
Promoting Decision Flows	34
Activating Flows	41
Managing Global Variables	45
Set an Event Time-Out	47
Audit Services	50
Data Collection for Performance Analysis	54
Chapter 4 • Environments and Resources	59
Repositories	60
System Resources	66
Library Resources	78
Databases	80

Chapter 5 • Decision Services Activities	83
Overview	84
SAS Activities	85
Web Service Activities	105
SAS Customer Experience Real-Time Server Engine	
Integration Activity	107
General I/O Activities	109
Guidelines for Creating Activities	117
Chapter 6 • Integration	119
Web Service Integration	119
WS-Security Integration	123
Integration with SAS Model Manager	132
REST API	135
Chapter 7 • Installation	137
Overview	138
Dependent SAS Products	139
Best Practices for SAS Decision Services Deployment Scenarios	140
Configuring SAS Decision Services	149
Deploying and Starting SAS Decision Services Web	
Applications in a Cluster	149
Rebuilding SAS Decision Services Design, Engine,	
and Monitor Server Web Applications	154
Rebuilding the SAS Web Application Server Configurations	155
Post-Installation Reconfiguration	155
Chapter 8 • Migration	157
Overview	158
Migration from SAS Real-Time Decision Manager 5.4	159
Migration from SAS Decision Services 5.5	159
Migration from SAS Decision Services 5.5M1	160
Migration from SAS Decision Services 5.6	161
Migration from SAS Decision Services 6.3	162
Migration from SAS Decision Services 6.4	163
Migrate Single DATA Step SAS Code	163

SAS Decision Services Utilities	164
Migrate Multiple DATA Step Code	167
Change Host Migration	169
Password Updates	171
Migrate Monitor Database to Stand-alone PostgreSQL	172
Chapter 9 • Best Practices	175
SAS Server Options	176
JDBC Performance Tuning	176
HTTP Client Code Usage	183
Use of SAS Data Sets	186
Initialize DS2 Activity Variables	187
Using SQLSTMT in Real-Time Applications	187
Suggested Initial Values for Key Tuning Elements in Systems with Heavy Workloads	193
Incorporating Pool Diagnostics	196
Chapter 10 • Custom Configuration	201
Standard System Resources	202
Configuring Additional Databases	203
Configuring Additional SAS Federation Servers to Form a Server Cluster	205
Changing the Database Selection	208
Configure Access to SAS Data Sets	212
Moving DS2 Persistence to a Database Management System ...	212
Moving Application Data from a Web Infrastructure Data Server to Other Databases	214

Appendix 1 • Database Requirements	217
Appendix 2 • Logs and Troubleshooting	219
Appendix 3 • Batch Execution	223
Appendix 4 • Activate Flows Using BatchActivator	241
Appendix 5 • REST API for SAS Decision Services Transaction Processing	245
Overview	246
Latency Requirements	247
Use Cases	247
Other Important Features	248
Terminology	248
Pagination	252
Media Types	254
Resources	268
Appendix 6 • SAS Decision Services Administration API	275
Overview	275
Use Cases	276
Terminology	277
Media Types	277
Resources and Collections	295
Recommended Reading	321
Glossary	323
Index	327

What's New

What's New in SAS Decision Services 6.4

Overview

SAS Decision Services 6.4 has the following enhancements:

- Broader Range of Deployment Options Available for Metadata Changes
- New Administrative API
- DS2 Activities That Can Call External RESTful Web Services
- SAS Decision Services HQ Plug-in for SAS Environment Manager

Broader Range of Deployment Options Available for Metadata Changes

VMware vFabric GemFire has replaced JGroups as the mechanism used to communicate metadata changes across the servers in a Decision Services engine cluster. As a result, User Datagram Protocol (UDP) multi-cast is no longer used, allowing for a broader range of deployment options.

New Administrative API

A new Administrative API has been added to enable client solutions to automate processes such as asset promotion and flow activation.

DS2 Activities Are Capable of Calling External RESTful Web Services

DS2 activities now have the capability to call external Representational State Transfer (RESTful) web services. The `tap_table` data type has been enhanced to support this feature, with the ability to encode data in Javascript Object Notation (JSON) form.

SAS Decision Services HQ Plug-in for SAS Environment Manager

The SAS Decision Services HQ Plug-in for SAS Environment Manager provides enhanced monitoring and control capabilities.

Accessibility

For information about the accessibility of any of the products mentioned in this document, see the usage documentation for that product.

Overview of SAS Decision Services

<i>What Is SAS Decision Services?</i>	1
<i>Decision Services Manager Plug-in for SAS Management Console</i>	2
<i>SAS Decision Services Repository</i>	4

What Is SAS Decision Services?

SAS Decision Services combines SAS analytics with business logic to deliver real-time decisions to workflow applications, complex event processors, or interactive customer channels. These channels include the web, mobile devices, call centers, point of sale (POS) locations, automated teller machines (ATMs), and others. The product provides an extensible and service-oriented architecture that supports high availability in environments requiring high-transaction volumes and low latencies.

An administrator performs the following tasks:

- controls which decision flows are in operation at any given time
- promotes decision flows from development to test to production environments
- configures and maintains the SAS Decision Services environments, ensuring that appropriate resources are available to meet performance requirements
- monitors and tunes the environments in which SAS Decision Services operates

- troubleshoots system issues using a variety of tools, such as performance logging and diagnostics

Decision Services Manager Plug-in for SAS Management Console

Most administrative functions are carried out using the Decision Services Manager plug-in. This plug-in is specifically designed for users who want to update, control, or monitor a design-time, test, or production Decision Services environment. The plug-in can be accessed from a Windows client machine that runs SAS Management Console. Users of this plug-in are system administrators, system operators, or performance analysts.

The plug-in is divided into two folders:

- The **SAS Decision Services servers** folder provides control of the SAS Decision Services environments, allowing an administrator to activate and deactivate decision flows and to change the values of global variables.

Each child of the SAS Decision Services servers plug-in folder represents a Decision Services design, test, or production environment.

A design environment consists of a design server, a SAS Federation Server, a SAS Workspace Server, and a content repository. Design environments are used by client solutions such as SAS Real-Time Decision Manager to author and unit test decision flows. The environments can also be used to run batch simulations.

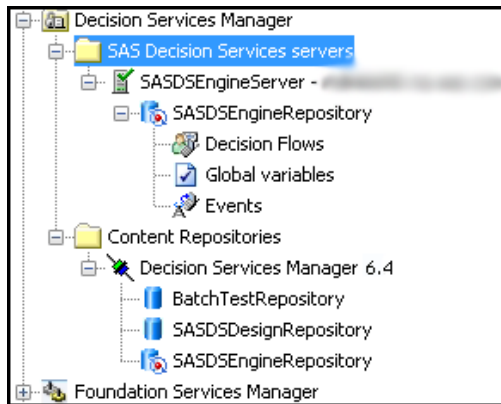
A test or production environment consists of a cluster of one or more engine servers, a cluster of one or more SAS Federation Servers, a SAS Workspace Server, and a content repository. Test environments are typically used for final testing of a decision flow before putting it into production. Production environments field requests and return decisions and recommendations to live channels, providing around-the-clock availability.

All environments typically include a third-party database management system that provides access to operational data.

Although an environment might contain multiple servers, each environment is managed by the plug-in as a single entity. For example, if you activate a flow in a particular environment, the flow is automatically activated on all of the servers that make up the environment.

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

Figure 1.1 *Decision Services Manager Plug-in Folders*



Icons represent the status of the real-time decision cluster. Here are the icons as well as the type of logical repository that they reference.



Indicates that the plug-in is connected to a running SAS Decision Services environment.



Indicates that the SAS Decision Services environment is not running.



Indicates a production repository.



Indicates a test repository.

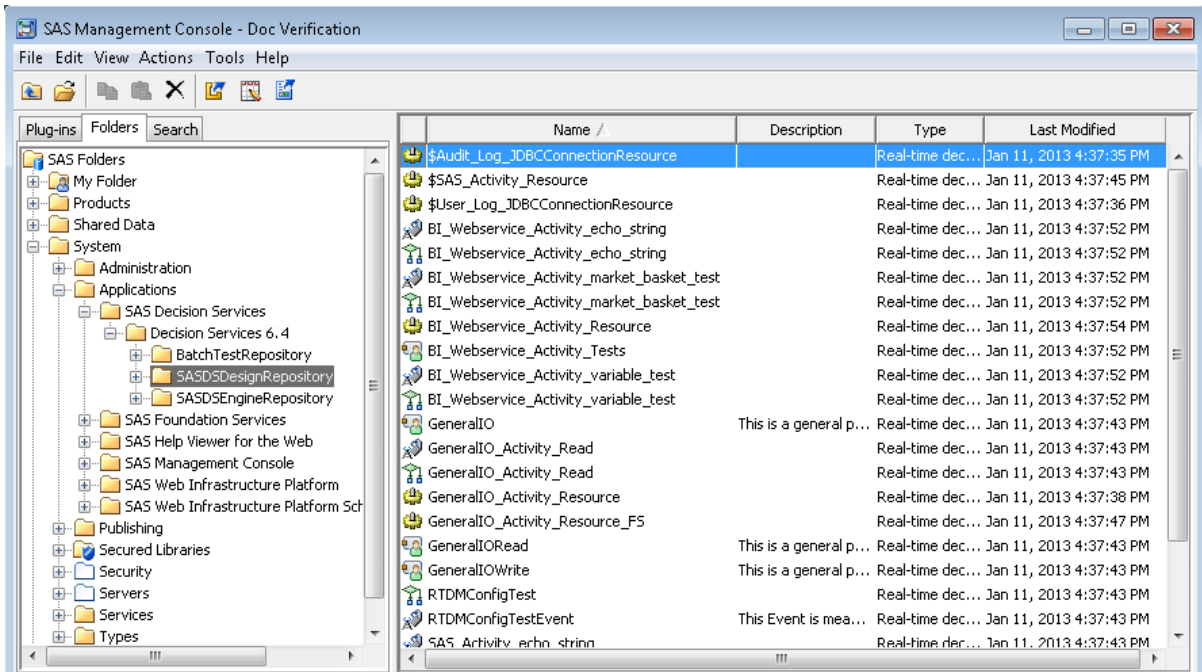


Indicates a development repository.

SAS Decision Services Repository

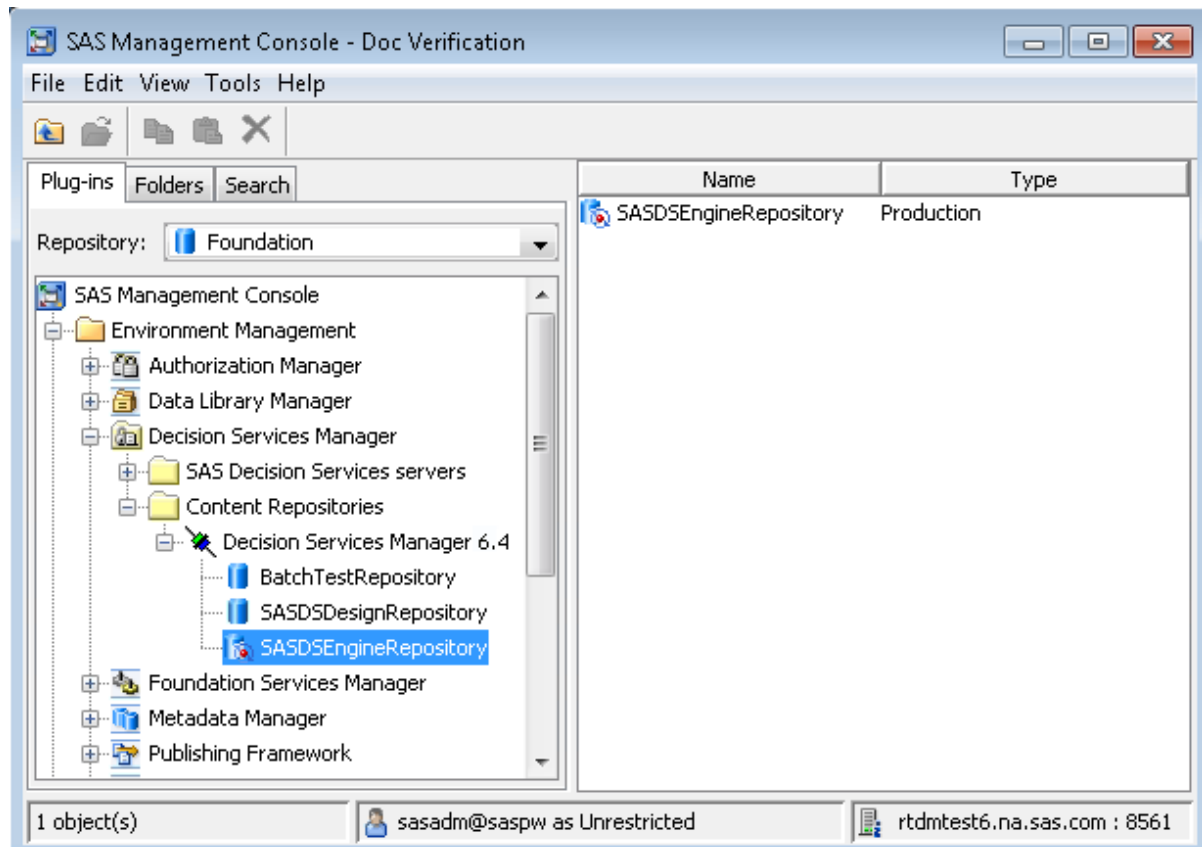
A SAS Decision Services content repository can be viewed in SAS Management Console by using either the **Folders** view or the Decision Services Manager plug-in. In the **Folders** view, all Decision Services artifacts are shown, and each has an associated name, description, type, and modification date.

Figure 1.2 Decision Services Manager Folders View



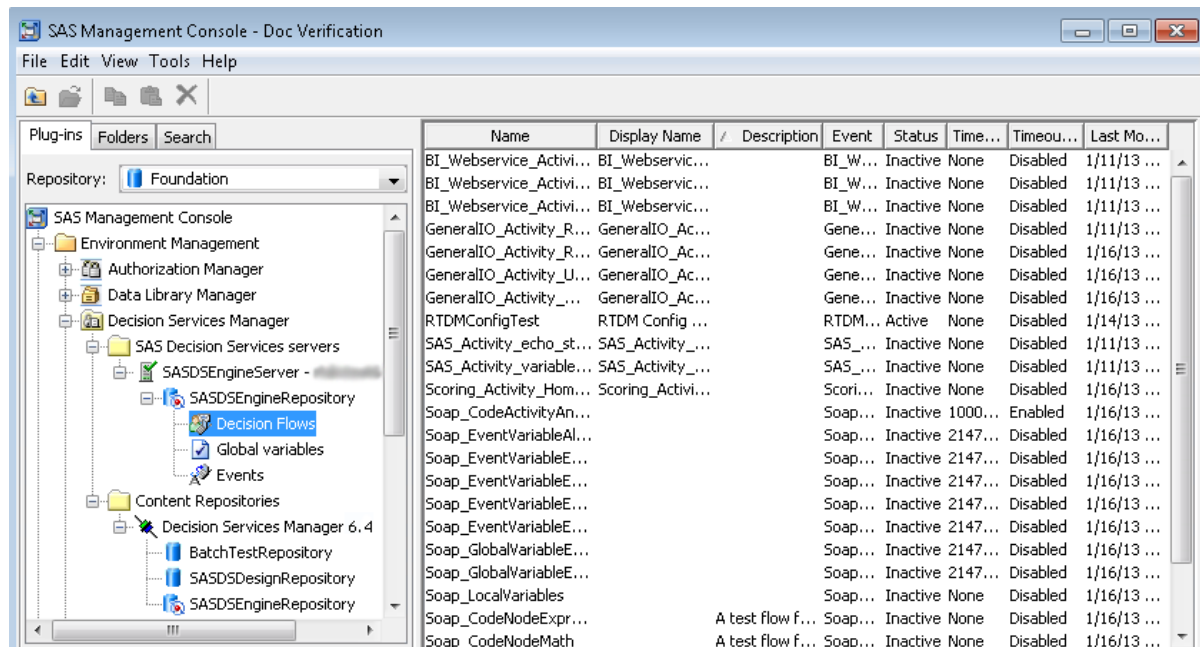
In the Decision Services Manager plug-in, the folder hierarchy is slightly different. It now shows a context-sensitive view of the repository and provides product-specific functionality. Only the artifact types that can be manipulated by the plug-in are displayed. By contrast, the **Folders** tab displays a non-context-sensitive view that works with any product. Although rendered differently, both options display the same data.

Figure 1.3 Decision Services Manager Plug-ins View



When SAS Decision Services 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 following display, **Decision Flows** is selected, and the flow name, display name, description, associated event, status (active or inactive), time-out value, time-out status (enabled or disabled), as well as the last modified date are displayed.



2

SAS Decision Services Concepts

<i>Modes of Execution</i>	8
<i>Scalability and Failover</i>	9
<i>Deployment Topology</i>	12
Overview	12
Production Environment	13
A Typical Configuration	13
Development Environment	14
Test and Production Environments	15
<i>Example: The Decision Flow</i>	16
<i>Life Cycle of a Decision Flow</i>	18
Overview	18
Development and Testing	18
Promotion	18
Activation	19
Monitoring	19
<i>Decision Flows, Building Blocks, and Artifacts</i>	20
Overview	20
Events	20
Activities	22
Decision Flows	22
Process Variables	23
System Resources	23

Library Resources	23
Global Variables	24
Sub-flow	24
Fault Response	24
Concurrent Execution of Nodes	25
ConcurrentWait Node	26
<i>Roles and Capabilities</i>	27
Overview	27
Managing Roles and Capabilities	27
Best Practices	27
<i>SAS Decision Services Monitoring</i>	28
Overview	28
Data Produced Per Engine	28
Data Collection	29
Persistence	30
Configuration	31
SAS Environment Manager	32

Modes of Execution

Batch execution capabilities have been added to SAS Decision Services, enabling consistent logic to be applied, regardless of whether the decision process is driven by a batch process or by a real-time channel.

In batch mode, transactions are read from a table and processed by the designated decision flow. Then results are written to another table. An input transactions table contains one record per transaction. Some or all of the columns of a transactions table must match the names and types of the corresponding event request variables. A transactions table might represent a subset or a superset of the event request variables. Likewise, some or all of the columns of a results table must match the names and types of the corresponding event reply variables. A results table might represent a subset or a superset of the event reply variables.

Batch execution proceeds as follows. A transaction is read from the input transactions table and the engine is called to process the transaction, which causes the appropriate decision flow to execute. The reply variables returned from the flow are written to the output results table. The process repeats until all transactions have been processed. Writing summary statistics is optional. Writing to the results table can also be suppressed. This feature is useful when a client application is interested only in hit count statistics.

Batch processing is divided into two categories:

- Design-time simulations — Run simulations on decision flows in order to measure behavior and distribution of results for a given set of transactions. The simulations interface is on the design server. Multiple users can run these design-time simulations concurrently without interfering with one another. The reason is because the design server runs each simulation in a private partition.
- Production batch jobs — Execute decisions in a batch in a test or production environment. Production batch jobs follow the same rules for flow and sub-flow activation as real-time execution. Production batch jobs are submitted through the Decision Services Monitor component.

Both simulations and batch jobs execute asynchronously, and both the design server and the monitor provide APIs to query status and progress. Both can also produce summary statistics. Producing summary statistics is optional.

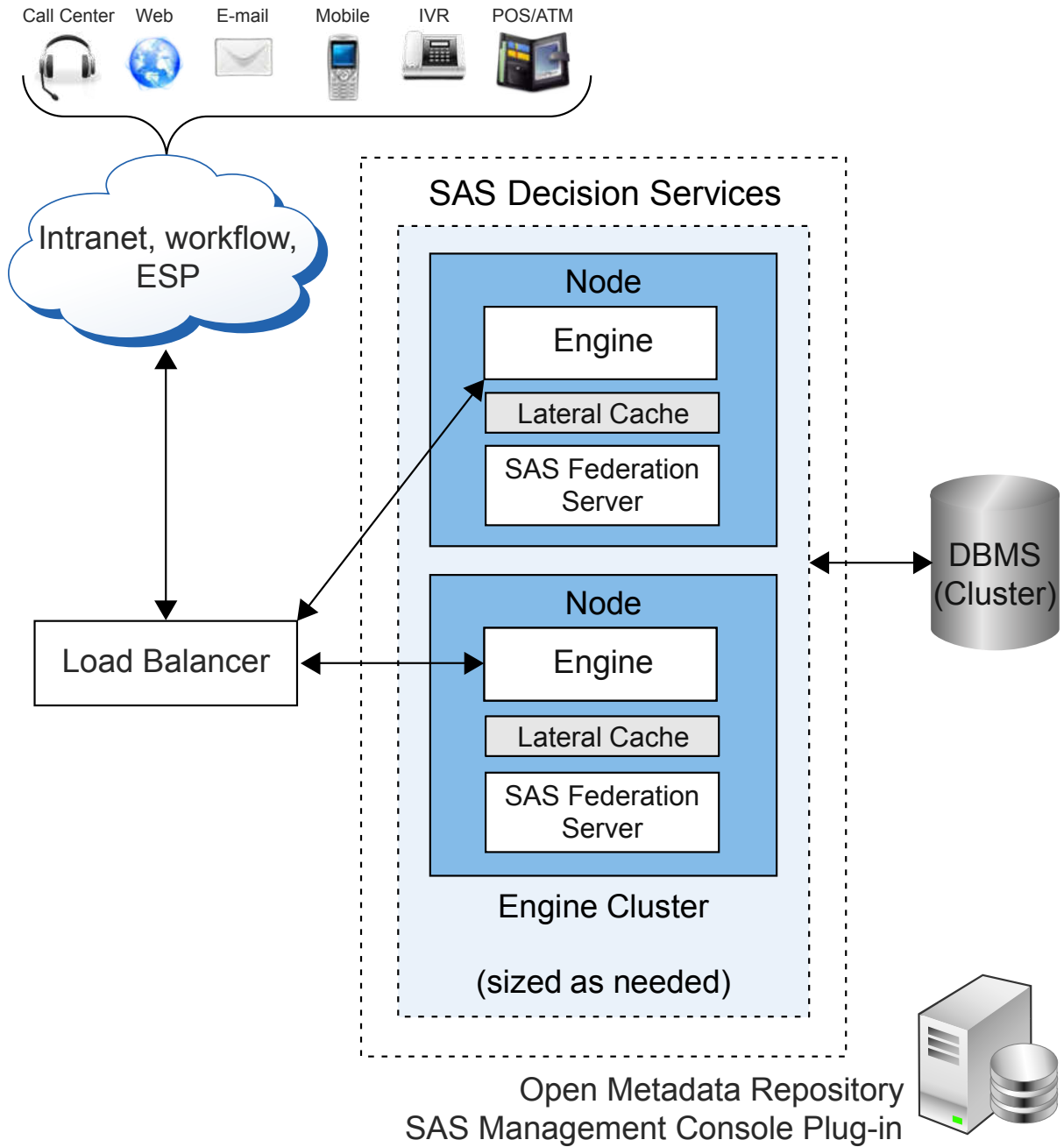
For more information about batch execution, see [Appendix 3, “Batch Execution,” on page 223](#).

Scalability and Failover

In SAS Decision Services, horizontal scalability and hardware failover are achieved through server clustering on multiple tiers. Vertical scalability and high performance are achieved by maximizing the parallel processing capabilities of the server hardware. The system is centrally managed using SAS Management Console.

The SAS Decision Services engine is deployed to SAS Web Application Server. The clustering and load-balancing capabilities of the server combine with the SAS Decision

Services threaded architecture to enable parallel execution. At any time, servers can be removed from or added to the cluster without stopping the application (for example, if a server fails and restarts). This operation is supported 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. This lateral cache is implemented using VMware vFabric GemFire, which does not require User Datagram Protocol (UDP) multicast. In addition to the middle tier, SAS Decision Services includes a configurable cluster of SAS Federation Servers.

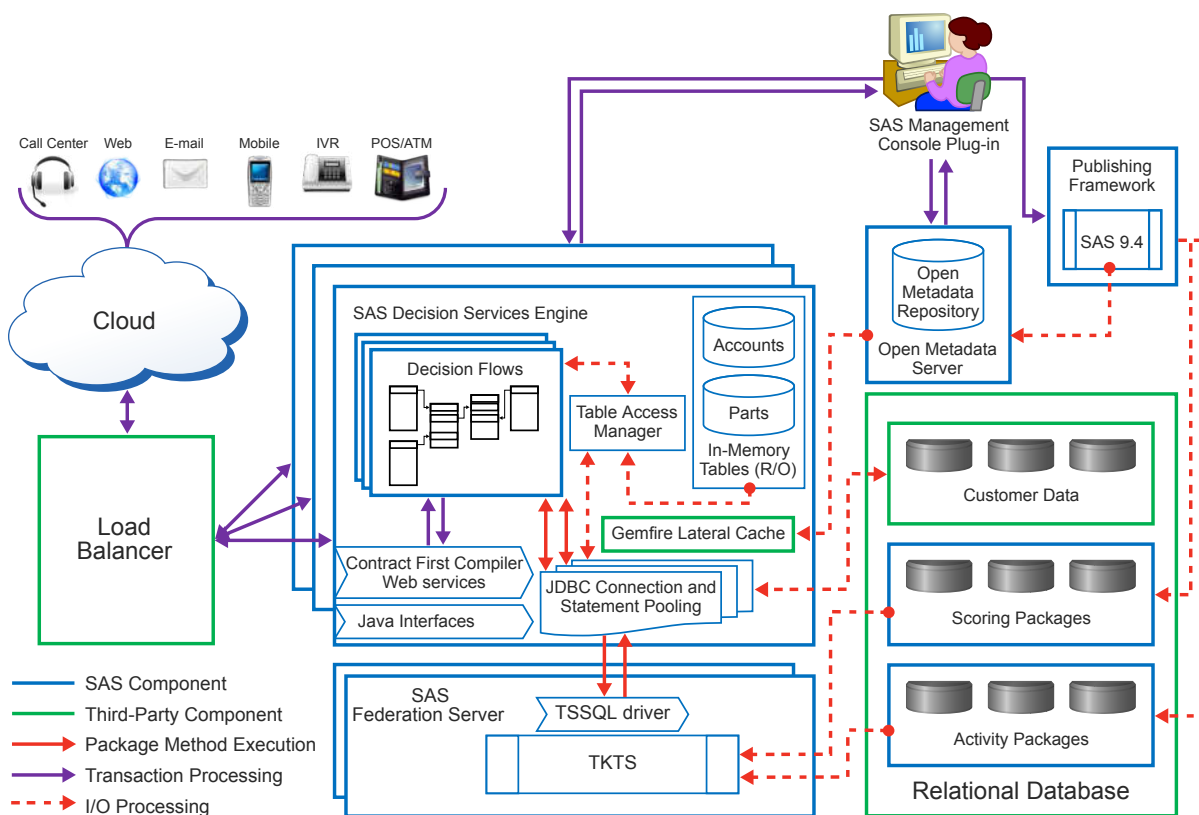
Figure 2.1 SAS Decision Services Run Time

Deployment Topology

Overview

The following figure shows the various logical components of SAS Decision Services when they are 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.

Figure 2.2 Logical Components in a Production Environment



Production Environment

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 flows.
- **SAS Decision Services Engine Servers** are configured in an application server cluster. These servers execute the decision flows that provide the real-time analytical decisions.
- **SAS Federation Servers** primarily run the SAS activities and score code that are based on DS2.
- **SAS Web Server** an HTTP server that is used to provide load balancing solutions for the real-time decision cluster enterprise. Using Service-Oriented Architecture (SOA) integration through web services, SAS Web Server is used as an integration point between external applications and a SAS Decision Services cluster. For more information, see the *SAS Intelligence Platform Middle-Tier Administrator's Guide*.
- **SAS Web Application Server** can be configured as a cluster and used for deployment of the SAS Decision Services engine server.
- **Database Servers** store data and DS2 packages, which implement SAS activity methods. SAS servers can be used to run BI web services for applications that require the execution of procedures or macro code.

The SAS Decision Services 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 process standards that reference internal approval. 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 that is 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.

SAS Management Console import and 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. Some client solutions follow a different promotion model, such as regenerating SAS Decision Services artifacts in each environment. See the documentation for your SAS solution before promoting your SAS Decision Services metadata artifacts.

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

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

Some client solutions might automate activation using the new Administrative API. See the documentation for your SAS solution before activating your SAS Decision Services metadata artifacts.

Development Environment

The development environment enables business users to create, test, edit, and delete decision flows. The SAS Decision Services design server provides this functionality through a web service API. Client applications provide user-friendly drag-and-drop interfaces, and use the SAS Decision Services 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 folder. Each repository folder resides in SAS Metadata Server. Repositories are managed by the Decision Services Manager plug-in.

A development environment consists of the following components:

- The client application's graphical user interface for building decision flows

- SAS Decision Services Design Server
- SAS Web Application Server
- SAS Federation Server and SAS Authentication Server
- SAS Metadata Repository
- SAS Management Console

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 continual operation, twenty-four hours a day, every day of the year.

As with the development environment, decision flows and their building blocks are stored in a repository. Repositories and their contents are managed by the Decision Services Manager plug-in or client application plug-ins. An important function of the Decision Services 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 operational channels or systems.

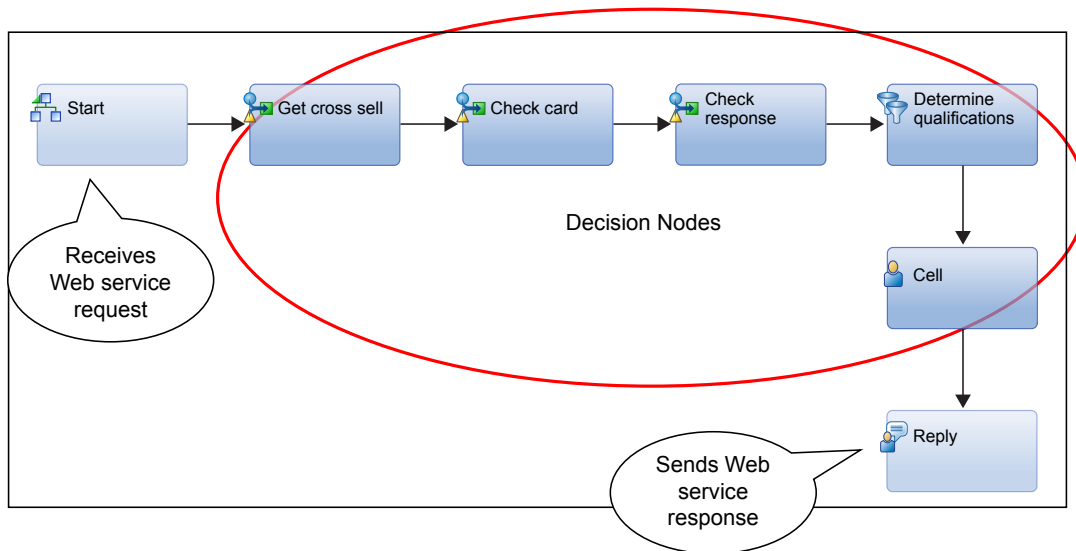
A test or production environment consists of the following components:

- SAS Decision Services engine server cluster and a load balancer
- SAS Web Application Server containing the engine server cluster
- One or more SAS Federation Servers
- SAS Metadata Repository
- SAS Management Console
- A third-party database management system

Example: The Decision Flow

Consider, for example, a retail business, where SAS Decision Services supports a website 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.

Figure 2.3 Scenario - Cross-sell Offer Example



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 Decision Services, requesting the best cross-sell offer to present.

Each active decision flow handles one web service request type. Therefore, 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 Decision Services, 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 that is 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 customer's primary purchase. This step could use 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 is not already in the customer's 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 customer's 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 might include branching rules, where the sequence of activity execution is controlled by a set of conditional expressions.

Life Cycle of a Decision Flow

Overview

To deploy a decision flow into production, it must be developed, tested, promoted to a production system, and activated. The following briefly examines each of these stages of the decision flow life cycle. Promotion and activation procedures are described in [“Promoting Decision Flows” on page 34](#).

Development and Testing

Users develop decision flows using the graphical user interface of a client application. This interface allows decision flows to be constructed by dragging and dropping activities from a palette. It also supports the development of decision flow tests.

A significant advantage of the activity model is that business users do not need to understand the complex algorithms used. Rather, they 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

SAS Decision Services deployment must include a development environment and a production environment. One or more test environments can be included as well. In this context, a test environment is just like a production environment except that it is not connected to live channels. The type of testing that is 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 Decision Services repository. If a flow is marked for deployment more than once, then the new copy of the flow overwrites any previous copy. When 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 a user marks a flow for deployment, the client application calls the SAS Decision Services 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. (For more information, see [“Promoting Decision Flows”](#).)

Note: Some client solutions automate the promotion process, making it unnecessary for the administrator to perform this function.

Activation

Each decision flow in a test or production environment is either active or inactive. Inactive flows are not loaded by a SAS Decision Services engine server. To put a flow into production (or to make it ready for testing in a test environment) the administrator must activate it. To remove a flow from production, the administrator deactivates it. For more information, see [“Activating Flows”](#).

Monitoring

The SAS Decision Services Monitor provides an API for querying activity hit counters and execution performance statistics. The Monitor also controls production batch execution, and provides access to batch job progress, status, and results. For more information, see [“SAS Decision Services Monitoring” on page 28](#).

Decision Flows, Building Blocks, and Artifacts

Overview

A set of activities and system resources is provided with the product and is typically configured by on-site SAS support personnel when your system is installed. On-site SAS support personnel can also work with your IT department to define the external events that are appropriate to your processing needs. Each external event defines the data that is exchanged between your system and SAS Decision Services for a specific interaction. An event typically consists of a set of request variables and a set of reply variables. The Decision Services Manager plug-in for SAS Management Console provides advanced functions that support the creation, editing, and deletion of system resources. (For more information, see [“Repositories” on page 60](#).) Other types of artifacts are created or deleted using the SAS Decision Services design server APIs. Client applications use SAS Decision Services design server APIs for this purpose.

Events

The SAS Decision Services Engine web service accepts SOAP messages called events. Each request for a decision is presented to the system as an event. These events and their associated decision flows are presented 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 can omit the reply message. An event makes up the contract between an external system and a decision flow, specifying the types of information that is contained within the request and reply. Typically, your IT department sets up your systems to make web service requests to the SAS Decision Services Engine, and on-site SAS support personnel define the events that map the data.

A response to an event is called an EventResponse. The XML payload for the event contains a name field, a header, and a body. The name field contains the name of the

event definition that is used to find the flow to execute. This header is distinct from the SOAP envelope header. The EventResponse also contains a header and a body.

The event header contains the following data items:

Identity

This is a string value that can be used to identify the event. The engine does not interpret the value of this field. However, it is logged in the engine log when there are faults or when trace logging is enabled. Although the engine does not enforce the uniqueness of this value, it is recommended that a unique value be provided for every call to track issues. This value is also returned as the value of the correlation ID for the EventResponse. The method `getEventIdentity()` returns the value of this input header element.

ClientTimeZoneId

This is a string value that contains the time zone ID of the client that is calling the engine. This value is used by certain SAS Decision Services functions to interpret date and time values that do not contain the time zone information. The valid values of this field are the time zone IDs that are supported by Java, and are based on the IANA time zone database. The method `getEventIdentity()` returns the value of this input header element.

SimulationDate

This is an optional element that has two attributes: `date`, an XML datetime, and `timeZoneId`, a string. The valid values for the time zone ID are the same as described in `ClientTimeZoneId`. If the `SimulationDate` element is not present, the default is the value of the `StartTime` element, returned in the event output header, plus the value of the input header element `ClientTimeZoneId`. The method `getEventSimulationDate()` returns a calendar that is constructed from these values.

The event response header contains the following data items:

CorrelationId

This is a string field that contains the value of the **Identity** field of the event.

StartTime

This is a timestamp that shows when the message was received by the engine.

CompletionTime

This is a timestamp that shows when the engine finished processing the event.

Body

The body contains data that is the input for, or output of, the engine when it is executing a specific event. The schema for this section is generic. Depending on the requirements of the EventDefinition, this section might contain zero or more data items that contain the input or output values.

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, INSERT, and UPDATE. Each action contains a set of inputs and outputs that are mapped to process variables. The activities that are provided with SAS Decision Services contain a rich set of functionality. The activities within a flow can execute sequentially or concurrently as specified by the containing flow.

SAS Decision Services functionality can be extended with custom activities. You can write a custom activity in the DS2 programming language, test it in a SAS session, and publish it to SAS Decision Services, where it can be included in decision flows.

SAS Decision Services stores DS2 source code in the activity metadata, using XML tags for DATA step and DS2 code that have been added to the activity schema. This feature enables the engine to automatically publish activity code as needed, guaranteeing referential integrity, and ensuring the decision services repository accurately represents the deployed code.

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 website or a call center, sends a request to SAS Decision Services. A decision flow includes 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.

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 web services. For example, many activities rely on running a SAS DS2 program to produce results. Because flows execute in SAS Web Application Server in the middle tier, these activities must communicate with SAS Federation Servers.

The fact that activities reference system resource information (rather than contain system resource information) makes flows portable between systems. SAS Decision Services supports configurable development, test, and production environments. Typically, the set of SAS Federation 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.

Library Resources

Library resources are special optional system resources that can assist database operations in certain circumstances. Library resources can hold an alias to a database schema name, allowing the alias name to be used to access tables within the schema. Library resources are optional and are not required for SAS Decision Services operation.

Global Variables

Global variables are used to control 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 redeploying the flow. For more information, see [“Managing Global Variables”](#).

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 an 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 that enables complex flows to 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 that 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 on how the parent flow is configured.

Fault Response

A fault response might be specified. This response is returned by the system whenever an error occurs. Given the high availability nature of the system, it halts processing only if a catastrophic error occurs that prevents every server in the engine cluster from functioning.

Sometimes actions that are performed in real time, such as sending a message to an operator, cannot be undone. Therefore, when an error occurs, real-time systems typically rely on compensation actions. In cases where a compensation action is not required in the event of an error, a predefined response might be returned to the caller.

Concurrent Execution of Nodes

Activity nodes and sub-flow nodes have an attribute that indicates whether they should be executed asynchronously. 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 in sequence. The order of execution of concurrent nodes is indeterminate.

There are three sub-tasks that take place in activity and sub-flow nodes. The sub-tasks occur in the following order:

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

If the nodes are marked concurrent, then step 2 is executed on a separate thread and the main thread continues processing the next node. Step 3, for the concurrent case, is performed when a ConcurrentWait node is reached by the main thread.

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 time-outs are also ignored. However, the node does execute. This technique can be used to execute a node concurrently when a response is not needed.”
- 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 value from the last concurrent node to finish will overwrite any values that were previously saved.

Note: This practice causes indeterminate behavior and is not recommended. Instead, direct the outputs of each concurrent node to separate process variables.

- 3 In case of an exception, such as a fault or time-out in any concurrent node preceding the ConcurrentWait node, no process variables are updated from that node.

ConcurrentWait Node

This node causes the main flow of execution to wait until all preceding concurrent nodes have finished execution. If a concurrent node throws an exception, the following ConcurrentWait node captures it and marks it as a fault. The wait in a ConcurrentWait node is timed. If a concurrent node does not complete execution in the allotted time, the following ConcurrentWait node produces a time-out 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.

ConcurrentWait nodes can be translated to DS2, but will run sequentially if sequential execution will yield equivalent results. If not, such flows cannot be translated to DS2.

To yield equivalent results, the flow must satisfy the following conditions for each node sequence starting with a concurrent node and ending with a ConcurrentWaitNode. Examples of concurrent node sequences include an activity method call or a sub-flow call where the node has the attribute “concurrent” set to True.

- The outputs of any concurrent node in the sequence cannot be used as input to any downstream node in sequence.
- The outputs of any non-concurrent node (such as an assignment node or code node) in the sequence cannot be used as input to downstream concurrent nodes in sequence.

Roles and Capabilities

Overview

SAS Decision Services users are assigned roles that enable them to perform specified actions, or capabilities, in the Decision Services Manager plug-in in SAS Management Console. One or more capabilities can be assigned to a role. For example, the Decision Services: Advanced role can contain capabilities such as viewing content XML, managing repositories, and purging data.

The following roles, with their assigned capabilities, are created during the installation and configuration of SAS Decision Services:

Decision Services: Administration

Provides edit, administrative, and delete capabilities.

Decision Services: Advanced

Provides advanced edit, administrative, and delete capabilities.

Managing Roles and Capabilities

If you have the appropriate permissions, you can create new users and groups and assign roles and capabilities in the SAS Management Console User Manager plug-in. To view or change the capabilities that have been assigned to a role, right-click the role name and select **Properties ► Capabilities**. SAS Decision Services capabilities are organized into folders. Expand a folder and select a capability to add it to a role.

Best Practices

You can create groups of users and then assign roles to the groups. The best practice is to assign roles to a group, rather than to individual users. You can also create new roles and assign capabilities to them, as well as edit the capabilities of existing roles.

SAS Decision Services Monitoring

Overview

The SAS Decision Services Monitor provides an API for querying activity hit counters and execution performance statistics. The Monitor also controls production batch execution, and provides access to batch job progress, status, and results.

The SAS Decision Services Monitor collects performance statistics from SAS Decision Services engines, saves them in a database, and supports querying this data. The following describes the implementation of the SAS Decision Services Monitor.

Data Produced Per Engine

SAS Decision Services engines expose the following statistics for monitoring:

- Event counts — The number of times an event is executed.
- Node counts — The number of times a particular node of a flow is executed.
- Flow response time —The average response time of a flow in milliseconds.

The monitoring framework monitors the SAS Decision Services engine, or a cluster of engines, collects the information over a specified duration, and support queries on the data.

Because monitoring data is aggregated across the servers of a cluster, there are finite limits on the granularity of data collection and the amount of data stored. These affect the accuracy of the queries. In general, the higher the accuracy, the more the storage and CPU cycles are required to collect the data.

Because there are multiple independent components in a SAS Decision Services engine cluster that might concurrently write to a database, only databases that support concurrent writes can be used to collect monitoring data.

While it is recommended that SAS Decision Services engines are installed on a cluster for high availability, the monitor also supports non-clustered deployments of SAS Decision Services engines.

Data Collection

Overview

The monitor polls all SAS Decision Services engines in the cluster to retrieve statistics on a regular basis. The data collection interval is configurable with reasonable restrictions. See the [“Configuration” on page 31](#) section for details about parameters that control the frequency of data collection. The engines continue to process data, whether the monitor retrieves data from it or not. Upon start-up, the monitor scans the Topology table for the list of engines running in a cluster. This table is scanned only at start-up. Any changes to the topology are not picked up by the monitor until the monitor web application is restarted.

Independent Threads

While it is not possible to guarantee a strict data collection interval, the system makes a best effort to do so. During data collection, the monitor communicates to the engines using spring remoting (Java serialization over HTTP). If an engine is down, the HTTP call will time out. Depending on network configuration, this might take time. To make the calls to the engines independent of each other, the monitor collects data from engines on separate threads.

Duration

The statistics that are retrieved represent an interval of time. The duration is determined by the time between the last retrieval from the engine (or the start of the engine) and the current time of retrieval. Any changes to event and node counts are captured. The average response time for flows is also captured. This means that if the monitor is stopped and restarted, the duration will be longer than usual and will include the counts during the time the monitor was not available. If an engine server crashes, accumulated counts since the last collection are lost. Therefore, using large data collection intervals can compromise the accuracy of the counts. On the other hand, more frequent collection uses more resources.

Starting and Stopping Data Collection

While the monitor constantly polls the engines for data, the engines do not collect data until a call to start data collection is received. When a call to start data collection is received by the monitor, it communicates this to one of the engines. A flag indicating whether data collection is enabled is held in the distributed cache accessible by all engines. The engine receiving the command updates this flag. All live engines periodically check this flag to determine whether it should start collecting data (or continue to do so) or stop collecting data. Engines that are not live can pick up the change in status when they come back up.

If data collection is enabled, the engines also scan the distributed cache for active flows and events, and then synchronize the counters for events and flow nodes to reflect the cache contents. A similar call is available to stop data collection. Not all applications might find the real-time counts useful. In such cases, the calling application should turn off data collection to reduce the load on the system.

The system can also be configured to start up and turn on data collection without explicitly requiring the command to turn it on. In earlier releases of SAS Decision Services, a system property could be set to true to configure the system to do so. This system property is now also available as a configuration parameter for the system.

Persistence

Overview

Data collected from engines are persisted in a relational database. In the event of the monitor process going down, historical data is retained.

Data Collected

As mentioned earlier, three types of data is collected: event counts, node counts, and flow response times. Changes to event and node counts for a duration are persisted in a record that includes the name of the event or the qualified name of the node (including the name of the flow), the count, engine information (address and port), and the start and end time of the duration. If the count for an event or node does not change in a duration, the record is not written.

Similarly, the average response time for a flow is also persisted in a record that includes the name of the flow, the average time in milliseconds, details of the engine, and the duration.

For activity method call nodes, metadata such as the name of the activity and the method called is also persisted. However, this information is not duration oriented and only the time at which this is retrieved is persisted.

Configuration

The system supports the following configuration items that control monitoring:

Item	Configuration Property	Definition
Query interval	<code>sasds.monitor.query.statistics.interval.seconds</code>	The interval in seconds when polling the engines for statistics. The default value for this parameter is 5 seconds.
Engine time out	<code>sasds.monitor.engine.query.timeout.seconds</code>	The maximum time to wait before the call to the engine to collect statistics is timed out. The default value for this parameter is 1 second.
Engine query thread pool parameters	<code>sasds.monitor.query.thread.pool.*</code> Note: The * indicates the inclusion of all of the properties that start with <code>dcsv.monitor.execution.thread.pool</code> .	The parameters for setting up the thread pool, for the threads that collect statistics from the engines in the cluster.
Data pruning enabled	<code>sasds.monitor.clearOldRecords</code>	If true, this will periodically clear old records. The default value is True.

Item	Configuration Property	Definition
Number of days of data to maintain	<code>sasds.monitor.keepRecordsForDays</code>	If <code>sasds.monitor.clearOldRecords</code> is True, then this deletes records older than the specified days from the current date. The default value is 30.
Continuous monitoring	<code>sasds.engine.data.collection.enableOnStartup</code>	If this condition is true, the monitoring will always be on.
Cache check frequency	<code>sasds.engine.data.collection.cache.check.milliseconds</code>	How frequently the system checks for changes in active flows in the cache, in milliseconds. The default value for this parameter is 1000.
Enable statistics collection when restarted	<code>sasds.engine.data.collection.enableOnStartup</code>	If this condition is true, then the system enables data collection whenever it is restarted without requiring an explicit start command. The default value for this parameter is False.

SAS Environment Manager

SAS Environment Manager can be used to monitor hit counters, flow performance, and system health information. For more information, see [“Using the SAS Decision Services HQ Plug-in for SAS Environment Manager” on page 54](#).

3

Common Operations

<i>Promoting Decision Flows</i>	34
Overview	34
Promotion Rules	34
Example: Promotion in SAS Management Console	35
<i>Activating Flows</i>	41
<i>Managing Global Variables</i>	45
<i>Set an Event Time-Out</i>	47
<i>Audit Services</i>	50
Overview	50
Events Logged	50
Data Logged	52
<i>Data Collection for Performance Analysis</i>	54
Using the SAS Decision Services HQ Plug-in for SAS Environment Manager	54

Promoting Decision Flows

Overview

Some SAS solutions that embed SAS Decision Services provide their own promotion frameworks, which automate the promotion, or activation, of Decision Services artifacts. Before following the promotion steps in this section, read the documentation provided with your SAS solution.

You typically promote a flow from a development environment to a test environment, or from a test environment to a production environment. (For more information, see [“Life Cycle of a Decision Flow” on page 18.](#)) However, flows and other artifacts can be promoted from any SAS Decision Services repository to any other SAS Decision Services repository. For more information about repositories, see [“SAS Decision Services Repository” on page 4.](#)

Promotion Rules

Note: During day-to-day operations, you typically need to promote only flows and variables.

- **As a general rule, resources should not be promoted.** System resources define how SAS Decision Services interacts with external systems. Because those systems and interactions are different in a production environment than in a development environment, promoting a resource can have undesirable consequences.
- **Activity promotion is necessary only after publishing a new or modified SAS activity.** When an activity is published, the source code for the activity is stored with the activity metadata. When a SAS activity is promoted, its source code is automatically promoted along with it. Be sure to promote a new or modified activity before promoting any flows that use it.
- **Do not overwrite an active flow.** If a flow, or other SAS Decision Services object, is promoted to an environment where an object with the same name and type already exists, the object in the target environment can be overwritten. When you overwrite

an active flow, the engine is not notified that the flow changed in the repository. Instead of overwriting the flow, deactivate the flow in the target system, promote it, and activate it. These steps cause the engine to load the updated flow. Note that when a flow is promoted, its state is automatically set to inactive.

SAS Decision Services includes a rich set of activities. If your organization develops a new activity that extends SAS Decision Services functionality, that activity must be promoted to each development, test, or production environment that uses the activity. Any system resources that are referenced by the new activity must also be created in these environments before flows that use the activity are activated.

Before promoting any updated activity where a method signature was modified, be sure to deactivate and delete all flows in the target repository that reference the original activity. Failure to do so might yield run-time errors or unexpected results.

When you define a new event (and create a corresponding web service request that calls SAS Decision Services), 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.

Example: Promotion in SAS Management Console

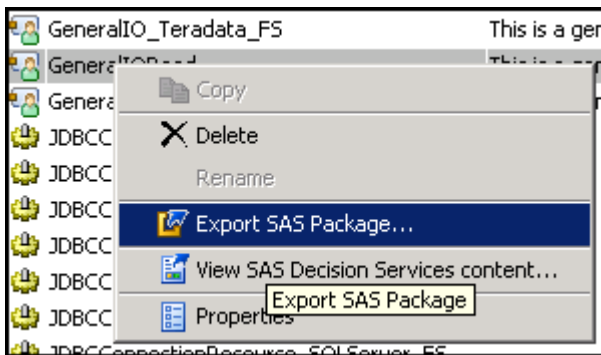
Promotion is accomplished in SAS Management Console by using the import and export functions from the **Folder** view. Promotion consists of exporting artifacts from one repository and importing them into another repository.

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

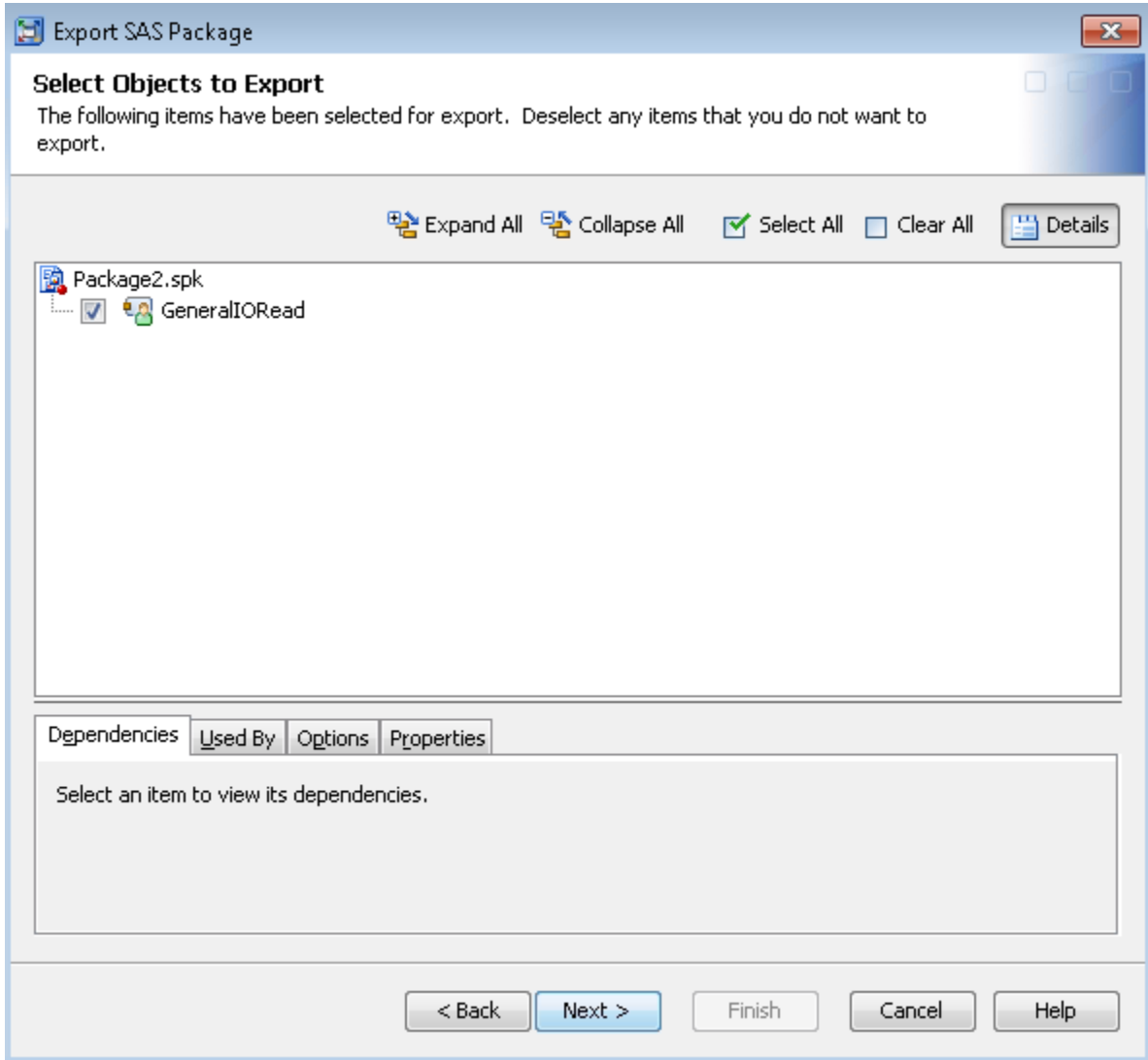
CAUTION! The Folder view in SAS Management Console does not restrict the locations to which artifacts can be exported. However, to avoid unpredictable results, always export from an individual artifact.

The following example illustrates the promotion of a flow from a development repository to a production repository. Although both repositories are contained by the same folder in the example, this condition is not required.

- 1 Launch SAS Management Console and click the **Folders** tab.
- 2 Expand the **System** and **Applications** folders.
- 3 Expand the **SAS Decision Services** and **Decision Services 6.4** folders.
- 4 Select **SASDSDesignRepository**.
- 5 Right-click the artifact that you want to promote (for example, GeneralIORead is the artifact shown below), and select **Export SAS Package** (note the previous caution).



- 6 Enter a package name, and click **Next**.
- 7 Select the artifacts that you want to promote. A convenient way to select only the boxes that you want is to select **Clear All**. Then select each XML file that you want to promote. Click **Next**.

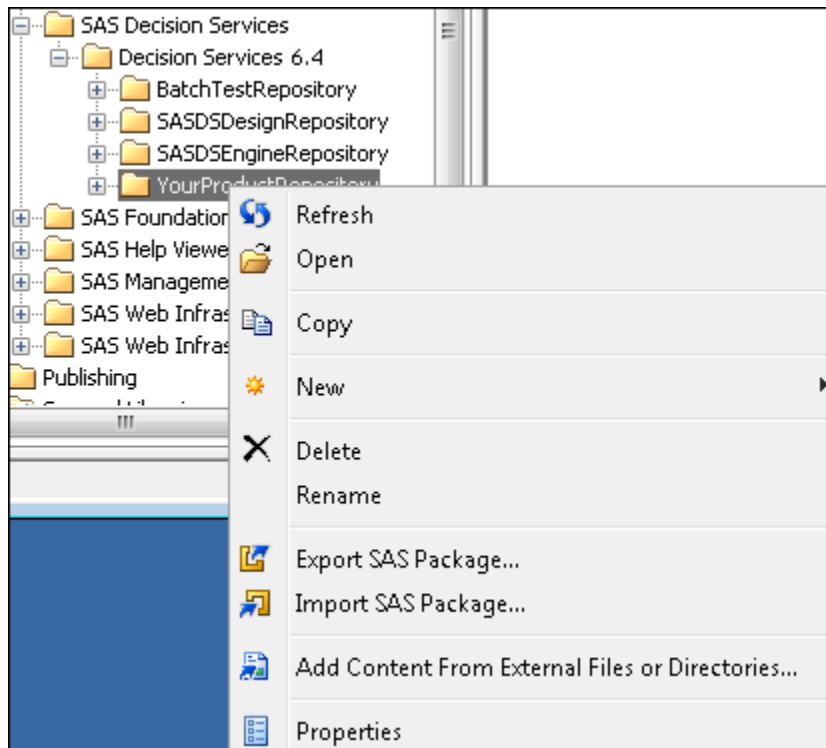


- 8 Verify the package name, location, and contents, and click **Next**.

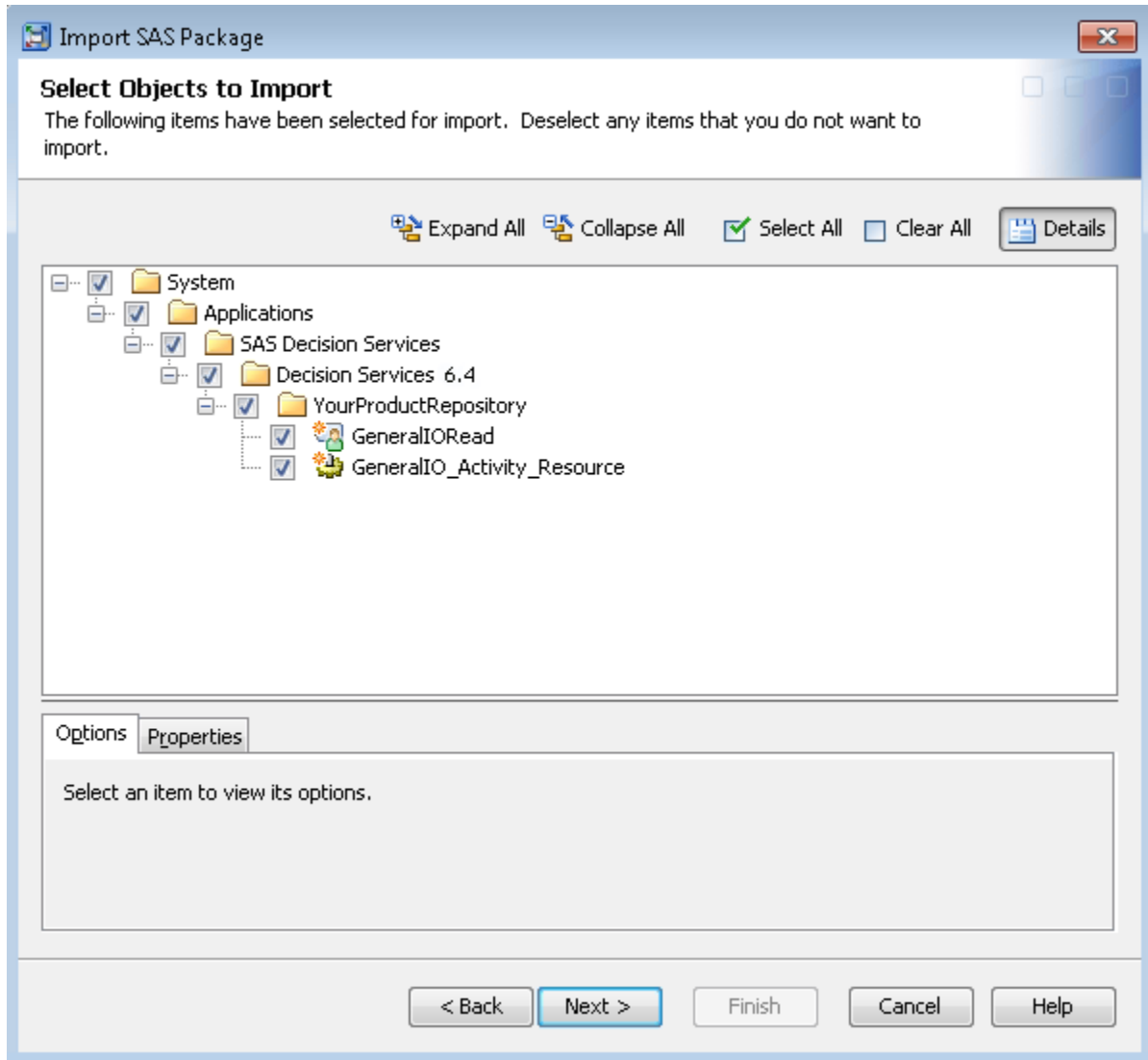
The flow has now been successfully exported from the development environment and saved in the package file called YourPackage.spk. The second part of the promotion process is to import the flow into the production environment.

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

CAUTION! The Folder view in SAS Management Console does not restrict the locations to which artifacts can be imported. To avoid unpredictable results, always import to a repository folder.



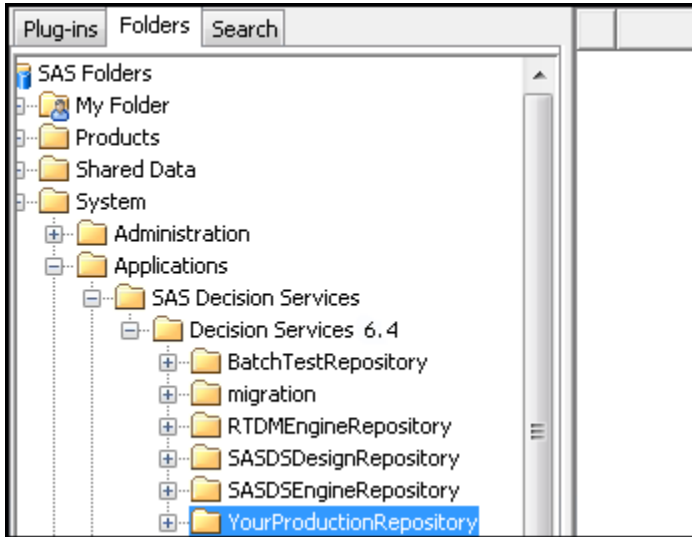
- 10** Navigate to your package name. If you import directly after exporting, then the package name is automatically supplied. To avoid overwriting existing artifacts, select **New Objects Only**. Click **Next**.
- 11** Verify that a check mark exists beside the XML file of each artifact that you selected. Click **Next**.



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

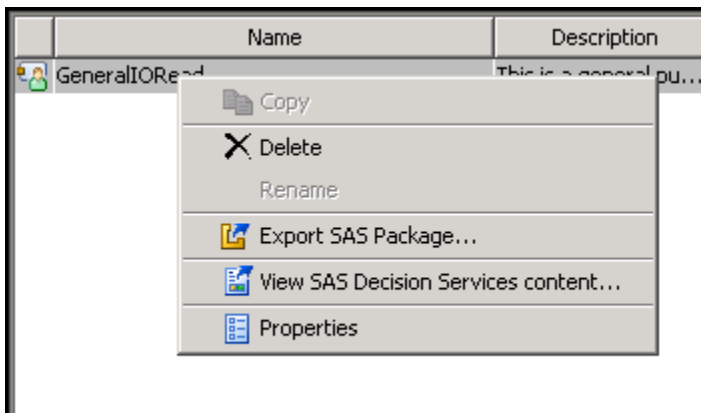
13 Click **Finish**.

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 as shown below.

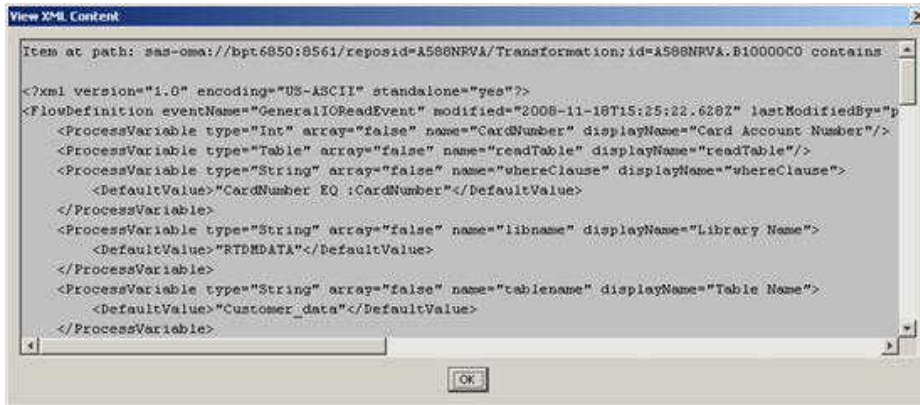


You can further verify that the promotion process was successful by viewing the contents of the XML file after promotion.

- 1 Click **YourProductionRepository** folder so that it appears in the right-hand pane.
- 2 Right-click **GeneralIORead** and select **View SAS Decision Services content**.



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



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

Activating Flows

Note: It is possible to automate the promotion and activation process. Activation can be performed through SAS Management Console, BatchActivator, Hyperic plug-in, or the SAS Decision Services Administration API. For batch activation, it is preferred that the SAS Decision Services Administration API be used. For more information, see [Appendix 6, “SAS Decision Services Administration API,” on page 275](#).

When a flow is activated, the 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` message.

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 they are referenced by an active flow. When loaded, flows and other artifacts are synchronized across the machines in the SAS Decision Services cluster and cached in memory for maximum performance.

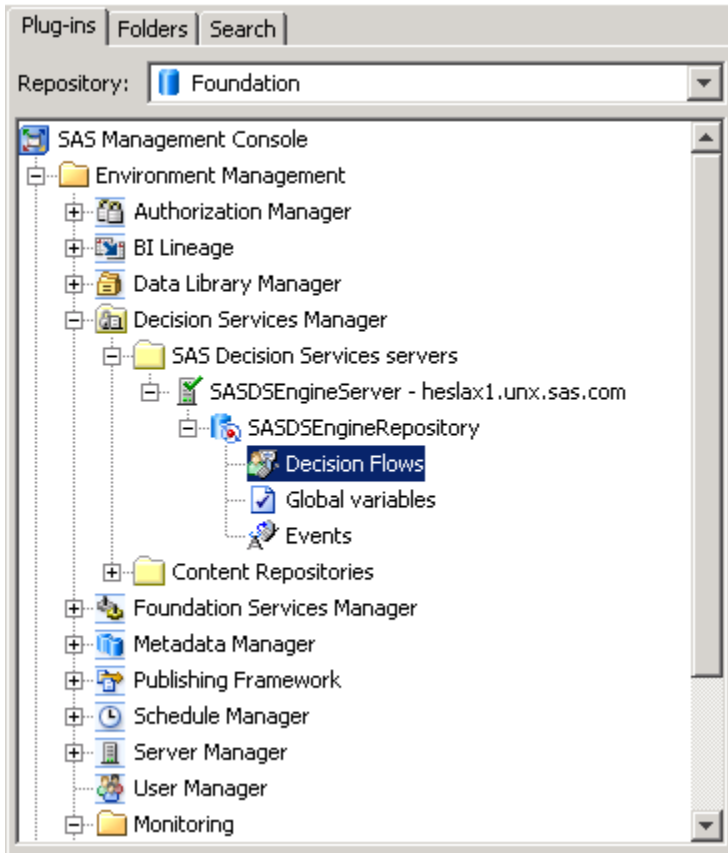
Each flow is bound to an event, which specifies the type of request a flow processes. Many different flows that reference the same event might exist in a repository, but only one of those flows can be active at any given time. For example, suppose flows A and B reference event X, and suppose A is active. Whenever event X is received, it is routed

to flow A. If you activate flow B, SAS Decision Services automatically deactivates flow A. Now, whenever event X is received, it is routed to flow B.

It is not necessary to activate or deactivate flows in the development environment. When a flow test is run, SAS Decision Services automatically loads, tests, and unloads the appropriate flow. Because the development environment is not connected to channels, the active or inactive states of the flows there are irrelevant.

To activate a flow:

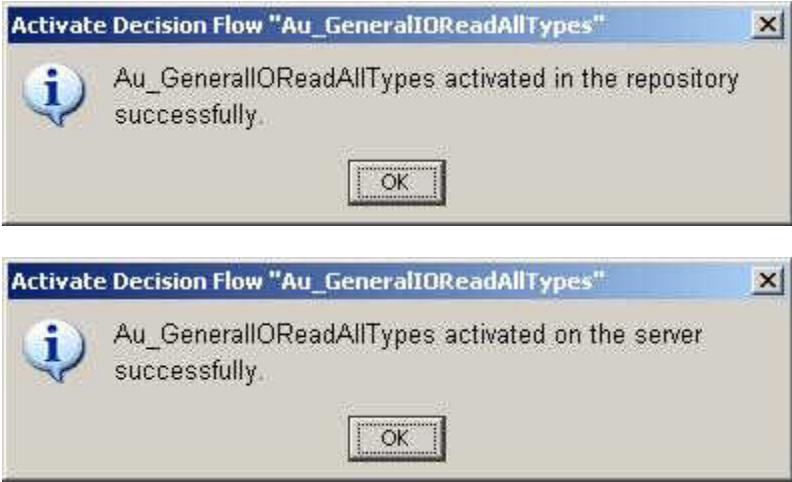
- 1 Launch SAS Management Console.
- 2 Expand **Decision Services Manager** and the **SAS Decision Services servers** folder.
- 3 Expand the SAS Decision Services system that contains the flow that you want to activate. In the example below, SASDSEngineServer represents a running engine that is deployed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



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

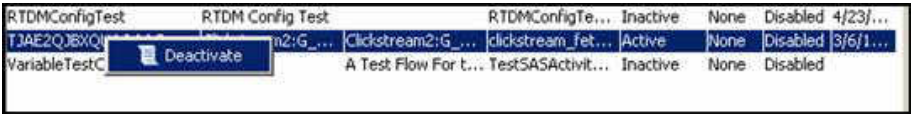
Au_GeneralIOInsert	Au_General IO Insert	Au_Generali... Inactive	None
Au_GeneralIOInsert_neo	Au_General IO Insert...	Au_Generali... Inactive	None
Au_GeneralIOInsert_oracle	Au_General IO Insert...	Au_Generali... Inactive	None
Au_GeneralIOReadAllTypes	Au_General IO Read All...	Au_Generali... Inactive	None
Au_GeneralIOUpdateAllTypesFlow...	General IO Update AllT...	Au_Generali... Inactive	None
Expression_FACT	Expression_FACT	AllTypes Inactive	None
RTDMConfigTest	RTDM Config Test	RTDMConfig... Active	None
Soap_AllInOneWithGlobal	Soap_AllInOneFlow	Soap_AllInO... Active	16000
Soap_ChainedSubflowCallFlow	Soap_ChainedSubflow...	Soap_Chain... Active	None
Soap_CodeActivityFlow	A Test Flow For the Te...	Soap_Code... Active	None
Soap_CodeNodeExpressions	A test flow for CodeNo...	Soap_Code... Active	None

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



Au_GeneralIOInsert_oracle	Au_General IO Insert...	Au_Generall... Inactive	None	Disabled	
Au_GeneralIOReadAllTypes	Au_General IO Read Al...	Au_Generall... Active	None	Disabled	
Au_GeneralIOUpdateAllTypesFlo...	General IO Update AllT...	Au_Generall... Inactive	None	Disabled	
Au_GeneralIOUpdateAllTypesFlo...	General IO Update AllT...	Au_Generall... 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**, as shown below.



Flows can be activated or deactivated in a system that is offline, to indicate which flows to load during system start-up. In this case, the green check mark on the engine icon is replaced by a red X, indicating the engine is not running. Upon successful activation, only the dialog box indicating successful activation or deactivation in the repository appears.

Flow activation and deactivation can also be scripted, allowing these operations to be controlled by workflow automation software. For a description of the scripting API, see [Appendix 4, “Activate Flows Using BatchActivator,” on page 241.](#)

Managing Global Variables

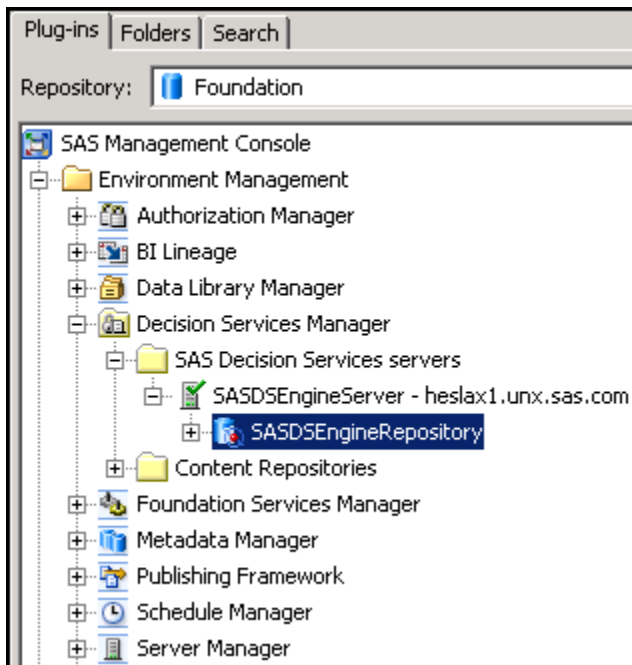
Global variables are threshold values that are used to tune the behavior of flows at execution time. Unlike process variables that are specific to a flow, the value of a global variable affects the behavior of every flow that references it.

For example, suppose a financial services institution wants 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.

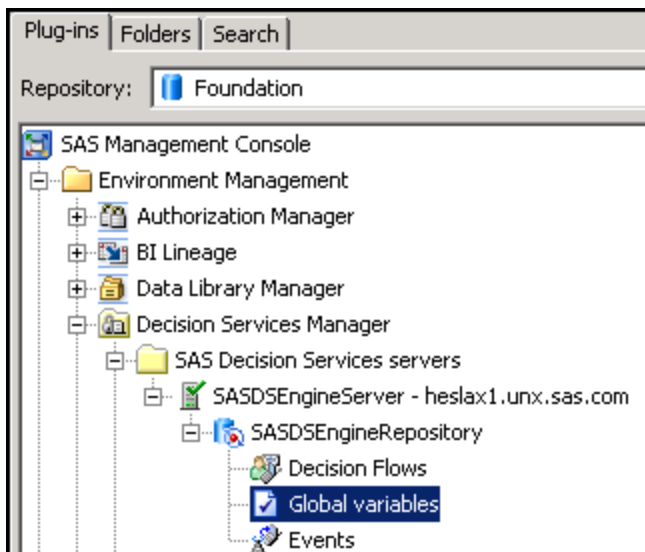
Global variables are created and assigned initial values when a flow that uses a global variable is designed. For security reasons, only an administrator whose role includes the `Set Global Value` capability can change the value of a global variable in a production environment.

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

- 1 Launch SAS Management Console.
- 2 On the **Plug-ins** tab, expand **Decision Services Manager** and the **SAS Decision Services servers** folder.



- 3 Expand the system that contains the global variable that you want to update. Expand the repository, and select **Global Variables**.



- 4 Right-click the global variable that you want to change, and click **Set Value**.

Soap_GVDatetime1	DateTime	'2007-07-1...
Soap_GVBoolean1	Boolean	true
Soap_GVIntArray1	IntegerArray	111;222;333
Soap_GVDatetime...	DateTimeAr...	'2007-07-1...
Soap_GVString1	String	'String1'
Soap_GVB...	BooleanArray	true;false;t...
Soap_GVFloatArray1	FloatArray	1.11;2.22;...

- 5 Enter the new value and click **OK**. Use either single or double quotation marks to indicate a string value.

The image shows a 'Set Value' dialog box with a title bar containing a close button. Inside the dialog, there is a label 'Value:' followed by a text input field containing the text 'String2'. At the bottom of the dialog, there are three buttons: 'OK', 'Cancel', and 'Help'.

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

When connected to online channels, the SAS Decision Services engine receives, processes, and responds to requests in real time. When defining an event in SAS Management Console, an administrator is able to specify a time-out setting for the event. Specifying a time-out setting controls the maximum amount of time that SAS Decision Services spends processing a request of that event type before returning a time-out error. It is possible for the flow that is associated with the event to also have a time-out setting. If that is the case, the flow time-out setting overrides the event time-out setting. This capability ensures that a response is provided within a specified time that is

appropriate for the channel and the type of customer interaction. If a request is not completed within the time-out interval, fault processing is initiated.

Time-out values can be set at three levels (from lowest to highest): system, event, and flow. Event and flow time-out values are optional.

- The system time-out value can be set during the installation and configuration of the SAS Decision Services design server and engine. If no value is specified by the user, then a default value is set. The value can be viewed in Configuration Manager in SAS Management Console. If you change the value through Configuration Manager, the new setting will not take effect until the system is restarted. Here are the property names for system time-out:
 - Engine server: `sasds.engine.execution.system.timeout.ms`
 - Design server: `sasds.designserver.test.system.timeout.ms`
- Use the Decision Services Manager plug-in of SAS Management Console to set the time-out value at the event level.
- The flow time-out value supersedes the event and system time-out values. It is set through the SAS solution used to design the decision flow. See the documentation that came with your SAS solution for details. For example, if your solution is SAS Real-Time Decision Manager, use the SAS Customer Intelligence Plug-in for SAS Management Console to set the time-out at the flow level.

Note: Specify all time-out values in milliseconds.

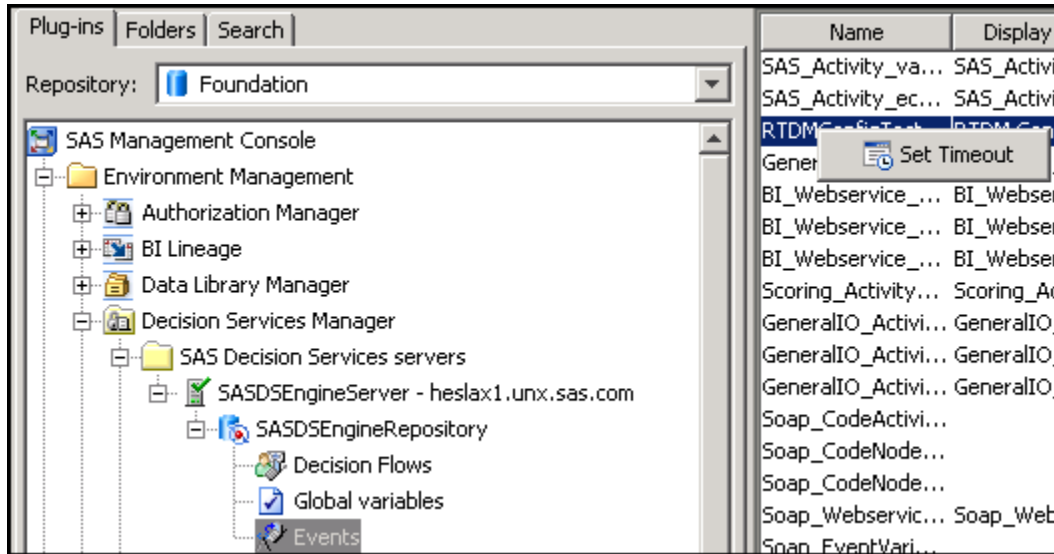
If a sub-flow is called, then the time-out value of the top-level flow or event is used. If the time-out values of the flow or the event are not specified, then the system time-out value is used.

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

- 1 Launch SAS Management Console.
- 2 Expand the **Decision Services Manager** and **SAS Decision Services servers** folders.

Expand the system that contains the event that you want to update. Expand the **Events** folder and the repository.

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



- 4 Select **Enable** to edit the time-out value, enter the value in milliseconds, and click **OK**. If **Enable** is cleared, then the time-out value for the event is disabled.



Audit Services

Overview

Auditing is an important part of the integration of SAS applications and solutions. In its simplest form, auditing is a feature that provides a historical record of user interaction with SAS applications, resources, and servers.

Events Logged

The following table lists the application events that are logged by different clients of the audit service. It is a two-step process to activate and deactivate flows and change the value of global variables. The first step is to change the status of the flow in the repository. The second step is to notify the engine to update the distributed cache. The first step generates the flow activation, flow deactivation, or variable value update events. The second step generates an engine update event, if the cache is updated.

Audit Event	Description	Clients
Engine shutdown	This audit event is logged when a SAS Decision Services engine is shut down.	

Engine update	<p>This audit event is logged when the distributed cache is updated as part of engine synchronization.</p> <p>Note: All calls to synchronize do not necessarily update the cache. For example, the cache is not updated if the contents of the repository are the same as the cache, or if there is a problem in publishing the DS2 code to SAS Federation Server. In such cases, no event is logged.</p>	This audit event is logged by the SAS Decision Services engine.
Flow activation	This audit event is logged when an administrator activates a decision flow. The audit event is logged after the status of the decision flow is updated in the repository.	This audit event is logged by the SAS Decision Services Administration API and the SAS Management Console Decision Services plug-in.
Flow deactivation	This audit event is logged when an administrator deactivates a decision flow. The audit event is logged after the status of the decision flow is updated in the repository. A flow might be deactivated because another flow that uses the same event is activated.	This audit event is logged by the SAS Decision Services Administration API and the SAS Management Console Decision Services plug-in.
Variable value updated	This audit event is logged when an administrator changes the value of a global variable. The audit event is logged after the status of the variable is updated in the repository.	This audit event is logged by the SAS Decision Services Administration API.

Consider the following items when viewing the log:

- Only the engine shutdown is logged.
- The SAS Management Console Decision Services plug-in does not log the change in the global variable values.
- The SAS Management Console Decision Services plug-in allows you to roll back the activation or deactivation of a flow when it cannot notify the engine to synchronize the cache. However, in such a case, it does not record the rollback in the audit log.
- No call to the audit log is made when a scripted activation or deactivation of a decision flow is made using Batch Activator.

Data Logged

The Web Infrastructure Platform (WIP) audit service logs the audit event data in two tables in the WIP data server. They are `public.sas_audit` and `public.sas_audit_entry`.

The primary table is `sas_audit`. Every row of that table corresponds to an audit event. SAS Decision Services generates an audit event whenever a SAS Decision Services event, flow, activity, or global variable is created, updated, or deleted. When an audit event occurs, SAS Decision Services fills in the following fields of the audit record:

Field	Description
<code>sas_audit.audit_id</code>	The unique ID of the record. It is also the foreign key to the <code>sas_audit_entry</code> table.
<code>sas_audit.object_type_id</code>	It contains the type ID of the object that was modified. Supported values are 118 (engine), 120 (decision flow), and 122 (variable).
<code>sas_audit.object_id</code>	The name of the object that is affected. This is typically the name of the decision flow that is activated or deactivated, the name of the variable whose value is changed, or the name of the engine when a cache update is logged.

<code>sas_audit.user_id</code>	The ID of the user who changed the object. This is significant only for the activation or deactivation of a flow or the change of the value of a variable. No user ID is logged when an engine synchronizes the cache.
<code>sas_audit.action_type_id</code>	The ID that indicates or codifies the type of changes to the system. The allowed values are 1 (variable update), 14 (engine shutdown), 36 (flow activation), 37 (flow deactivation), and 43 (engine synchronization).
<code>sas_audit.timestamp_dttm</code>	The timestamp of when the event took place.
<code>sas_audit.audit_info</code>	Text describing the event.
<code>sas_audit.executor_nm</code>	The name of the application that caused the change. The values are Decision Services Administration API, Decision Services Engine Server, or SAS Management Console.

The `sas_audit_entry` table contains the additional information for certain types of audit events, such as engine synchronization and variable value update. It contains two fields that are significant to SAS Decision Services: `sas_audit_entry.property_nm` and `sas_audit_entry.new_value_txt`. These fields are used in conjunction to describe additional information about the event.

Here are the possible values for engine synchronization:

- `property_nm` contains `RTDMFlow`, and `new_value_txt` contains the name of the flow that is loaded in the cache.
- `property_nm` contains `RTDMVariable`, and `new_value_txt` contains the name of the variable that is loaded in the cache.
- `property_nm` contains the name of the global variable that is loaded in the cache, and `new_value_txt` contains the value of the variable.

For array type variables, there are multiple rows, each containing one array element value.

For variable value updates, `property_nm` contains value, and `new_value_txt` contains the value of the variable. For array type variables, there are multiple rows, each containing one array element value.

Data Collection for Performance Analysis

Using the SAS Decision Services HQ Plug-in for SAS Environment Manager

Overview

Using the SAS Decision Services HQ Plug-in for SAS Environment Manager, you can collect engine performance data, such as event hit counts, node hit counts, and average flow response time. The plug-in also provides system health information. Every tracking service is displayed as an individual service under SAS Decision Services Monitoring Server. You can also find physical memory size, virtual memory size, and CPU usage percentage information.

In order for SAS Environment Manager to access the SAS Decision Services monitoring plug-in, a SAS Environment Manager agent needs to be installed on each server in a SAS Decision Services engine cluster. The auto-discovery process might take longer than 30 minutes, depending on the background process of the SAS Environment Manager server. After the SAS Decision Services monitoring plug-in is in the inventory of the SAS Environment Manager server, you can select the SAS Decision Services resource that is linked to the SAS Decision Services monitoring server, in order to collect the engine performance data.

For more information about SAS Environment Manager, see SAS Environment Manager User's Guide at <http://support.sas.com/documentation/onlinedoc/sev/index.html>.

Monitoring Service Inventory

To view the overall monitoring services, click **Inventory** on the SAS Decision Services Monitoring Server screen. The current monitoring services are listed under **Services**. The total number of services, as well as a list of services by type, is displayed at the top

of the **Service** section. This gives you a good sense of what events, nodes, and flows are being monitored.

You can click the name of any of the services in the list to find general information about that service.

Dashboard
Resources
Analyze
Administration
Manage

Browse
AIMVM09 Decision Services Monitoring Server -- Experimental
Return to AIMVM09 Decision Services Monitoring Server -- Experimental

Description: Decision Services Monitoring S...
Owner: HQ Administrator () - Change...

Max per-process memory (KB) : Unlimited
Max user processes : Unlimited
Number of Events : 0

Max stack size (KB) : 1,020
Max virtual memory (KB) : 2,097,152
Number of Flows : 0

Map
Tools Menu

Monitor
Inventory
Alert
Control
Views

General Properties

Description: Decision Services Monitoring Server for AIMVM09
Date Created: 01/07/2014 04:57 PM

Date Modified: 01/07/2014 04:57 PM

Resource Type: Decision Services Monitoring Server 6.3
Modified By: HQ Administrator ()

Edit

Type & Host Properties

Install Path: /DecisionServices
Host Platform: .na.SAS.com

Edit

Services

Total Services: 0

Services By Type:

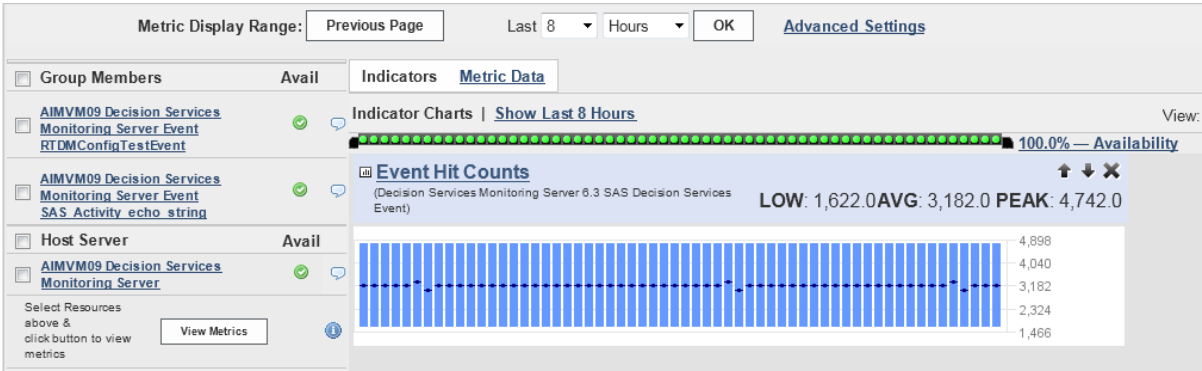
Service
Service Type
Description
Availability

Delete
Total: 0
Items Per Page: 15

Event Hit Counts

When an event is monitored, the SAS Decision Services monitoring plug-in periodically receives hit counts from the monitoring server. The hit counts for each event are accumulated from the time the SAS Decision Services server is started. The default tracking interval is set to one minute. The tracking interval can be changed. For more information, see “[Configuring the Monitoring Interval](#)” on page 57.

Under **Monitor**, click **Decision Services Monitoring Server 6.4 Decision Services Event** to see the total hit counts indicator chart for all events. To see the hit counts for an individual event, click the event listed under **Group Members**. To see a detailed tracking chart, click **Event Hit Counts** located above the indicator chart. Click **Metric Data** to view the current monitoring numeric data table.



Node Hit Counts

Node hit counts work in the same way as event hit counts. When a node is monitored, the SAS Decision Services monitoring plug-in periodically receives hit counts from the monitoring server. Also, like an event hit count, the hit counts for each node are accumulated from the time the SAS Decision Services server is started. The default tracking interval is set to one minute. The tracking interval can be changed. For more information, see “[Configuring the Monitoring Interval](#)” on page 57.

Under **Monitor**, click **Decision Services Monitoring Server 6.4 Decision Services Node** to see the total hit counts indicator chart for all nodes. To see the hit counts for an individual node, click the node that is listed under **Group Members**. To see a detailed tracking chart, click **Node Hit Counts** located above the indicator chart. Click **Metric Data** to view the current monitoring numeric data table.

Average Flow Response Time

When a flow is monitored, the SAS Decision Services monitoring plug-in periodically receives the average response time in mini-seconds from the monitoring server. The average response time for each flow is accumulated from the time the SAS Decision Services server is started. The default tracking interval is set to one minute. The tracking interval can be changed. For more information, see [“Configuring the Monitoring Interval” on page 57](#).

Under **Monitor**, click **Decision Services Monitoring Server 6.4 Decision Services Flow** to see the average response time indicator chart for all flows. To see the average response time for an individual flow, click the flow listed under **Group Members**. To see a detailed tracking chart, click **Average Response Time** located above the indicator chart. Click **Metric Data** to view the current monitoring numeric data table.

Configuring the Monitoring Interval

The default tracking interval is set to one minute for all monitoring services. To change the interval, follow these steps:

- 1 Switch to Metric Data view by clicking **Metric Data** from any monitoring service.
- 2 Select the metric data in the first column of the table by clicking the check box.
- 3 Change the time interval in the text field by entering any integer.
- 4 Click **Go** to finalize the change.

Using the Control Actions

Click **Control** on the SAS Decision Services Monitoring Server screen to send control commands to the monitoring server. From the **Control Action** drop-down menu, select one of the following commands:

Check Data Collection Status

displays the current data collection status.

Remove All Realtime Data

cleans up all real-time data in the data collection table.

Enable Realtime Data Collection

enables statistics collection on the monitoring server.

Disable Realtime Data Collection

disables statistics collection on monitoring server.

All commands are sent through the web service call.

4

Environments and Resources

<i>Repositories</i>	60
Overview	60
About Repositories	60
Create a Repository	60
Delete a Repository	64
<i>System Resources</i>	66
Overview	66
JDBC Connection System Resources	67
Specify a New System Resource as a JDBC Connection	72
Specify a New System Resource as a Web Service Connection	76
Specify a New System Resource as an HTTP Connection	77
<i>Library Resources</i>	78
Overview	78
(Optional) Define a Schema Alias	78
<i>Databases</i>	80
Overview	80
Connection Information for the JDBC Data Source	80

Repositories

Overview

Before using any of the advanced SAS Decision Services functions, such as creating a new repository, make sure that you understand how to administer content repositories.

Use the advanced functions in the Decision Services Manager plug-in to SAS Management Console to create and delete repositories.

About Repositories

Repositories 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.

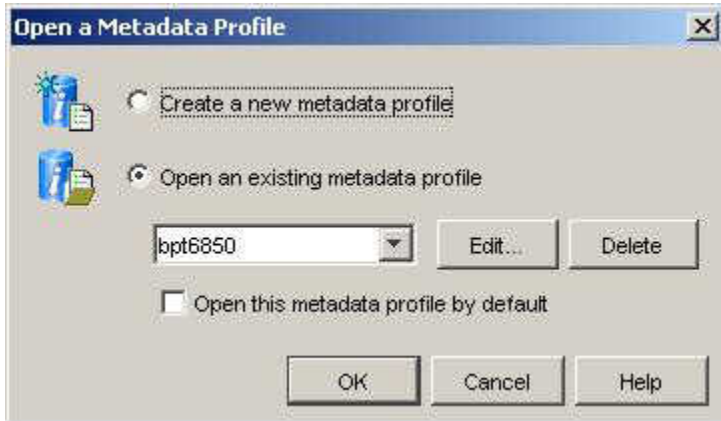
A repository does not have to be associated with a server; it can be used simply as a storage area for artifacts.

A repository resides in SAS Metadata Repository. However, each Decision Services development, test, and production environment maintains a repository where the artifacts of the environment are kept.

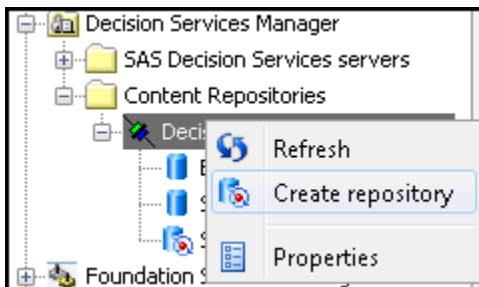
Create a Repository

To create a new SAS Decision Services repository, follow these steps:

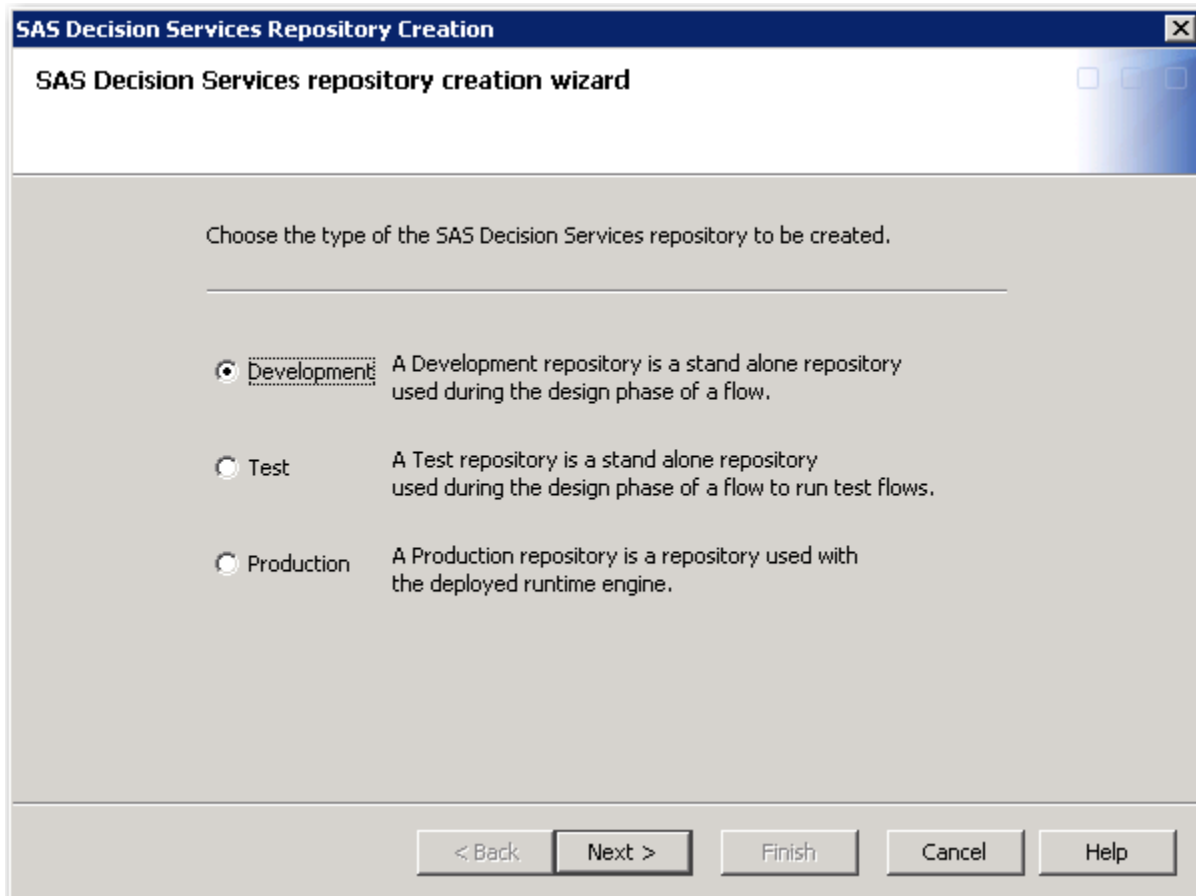
- 1 Log on to SAS Management Console. Select the metadata profile that is associated with the SAS Metadata Server where you want to create your repository. For more information about metadata profiles, see the SAS Management Console Help.



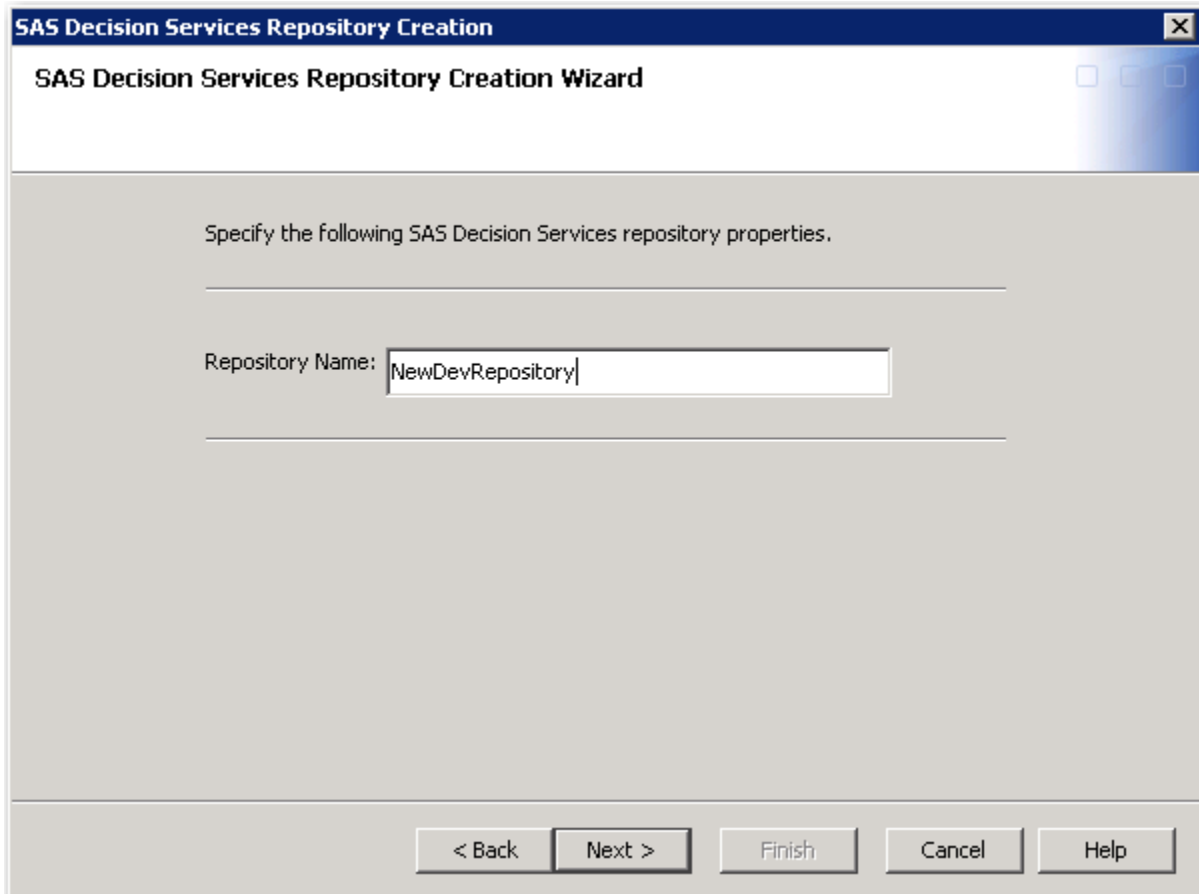
- 2** Expand **Decision Services Manager** and **Content Repositories**.
- 3** Right-click the folder where you want to create your repository, and select **Create repository**.



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



- 5 Enter a name for your new repository. The following example shows the creation of a new repository called NewDevRepository. Click **Next**.



The screenshot shows a Windows-style dialog box titled "SAS Decision Services Repository Creation Wizard". The title bar is dark blue with the text "SAS Decision Services Repository Creation" and a close button (X) on the right. Below the title bar, the text "SAS Decision Services Repository Creation Wizard" is displayed in a larger font. The main area of the dialog is light gray and contains the instruction "Specify the following SAS Decision Services repository properties." followed by a horizontal line. Below this line, the label "Repository Name:" is followed by a text input field containing the text "NewDevRepository". Another horizontal line is positioned below the input field. At the bottom of the dialog, there is a row of five buttons: "< Back", "Next >", "Finish", "Cancel", and "Help". The "Next >" button is highlighted with a darker border.

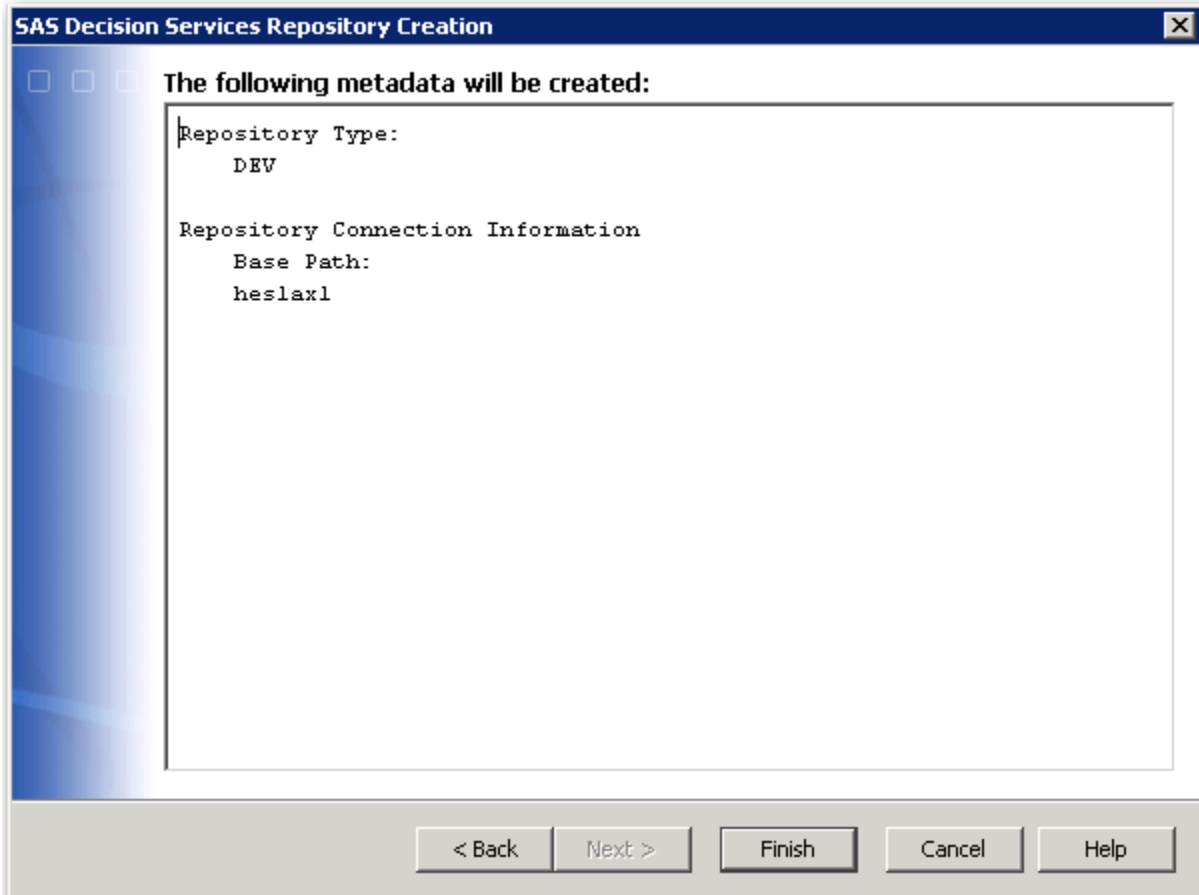
SAS Decision Services Repository Creation Wizard

Specify the following SAS Decision Services repository properties.

Repository Name:

< Back Next > Finish Cancel Help

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



7 Verify that your repository was created correctly by expanding your repository folder.

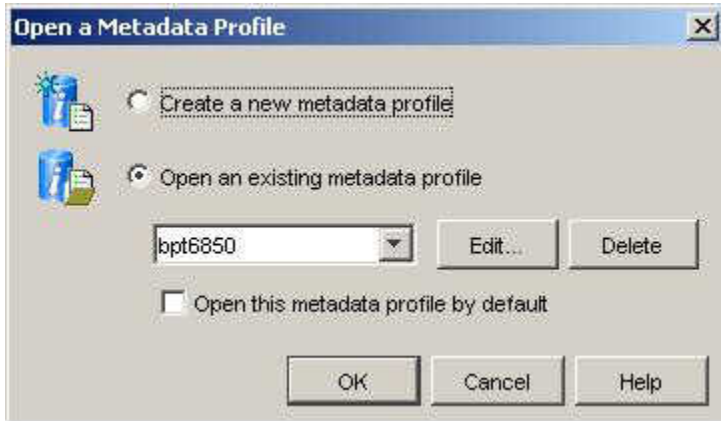
A repository is bound to an engine or design server when that server is installed and configured. For more information, see [Chapter 7, “Installation,” on page 137](#).

Delete a Repository

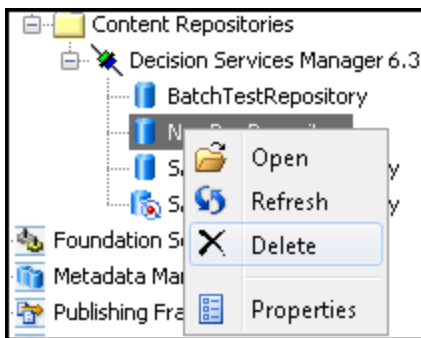
CAUTION! Deleting a repository is an irreversible operation.

To delete a repository:

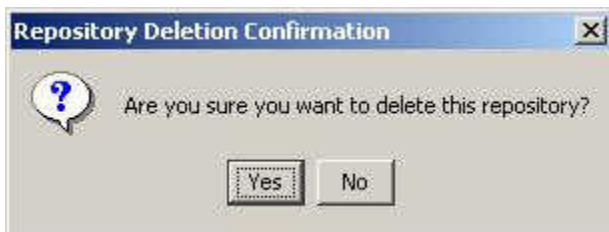
- 1 Log on to SAS Management Console. Choose the metadata profile that is associated with the SAS Metadata Server that contains the repository to delete.



- 2 Expand **Decision Services 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

Overview

System resources enable decision flows to access and interact with resources such as SAS servers, database servers, or external web services. Activities reference the system resources by name.

For example, many activities run a SAS DS2 program to produce results. The middle tier portion of these activities must communicate with a SAS Federation Server. A system resource type named JDBC Connection provides the information that is needed to facilitate such communications. More specifically, the JDBC Connection system resource contains information that is needed by a SAS activity to execute a DS2 program running on the SAS Federation Server.

Also, a different JDBC Connection system resource is used to connect to database servers for use in the General I/O activity. When you are accessing database tables other than SAS data sets, these resources point directly to the database using the database's native JDBC driver.

The web service system resource is often used to connect to external web services. By providing the end point URL, SAS Decision Services can use the web service that is pointed to.

The HTTP system resource is used for exchanging information between SAS Decision Services and SAS Customer Experience Analytics.

Activities use a name to reference system resources instead of containing the resource information directly. Thus, flows are 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 that have the same names. However, the

information that is contained by these system resources differs from environment to environment.

JDBC Connection System Resources

About JDBC Connection System Resources

JDBC Connection system resources are used by both SAS activities that execute DS2 programs and by General I/O activities that access database records. The basic fields are listed [in step 5 of the following section on page 73](#). In the case of General I/O, the Connection Options value is not required. For General I/O activities, multiple database server URLs can be specified, in a single system resource, in order to support database clusters that do not have server-side load balancing. Each space-separated URL references one node of a clustered database environment.

To connect to SAS DATA sets and to execute DS2 SAS activities, a JDBC Connection system resource must be configured to connect to one or more SAS Federation Servers. The JDBC Connection system resource named `$SAS_ACTIVITY_RESOURCE` is configured for this purpose.

Advanced options are available that allow for the fine tuning of the connection and statement pools used by SAS Decision Services. These values should be set to appropriate values based on the hardware being used. A list of these options appears in [“Tuning Controls” on page 176](#).

To allocate computing resources efficiently, set up more than one SAS Federation Server in the server tier. Every server within a given cluster processes the same activity set. The following example illustrates this concept.

Each middle-tier engine server can be configured to load balance every SAS Federation Server. Such a configuration guarantees that a middle-tier server failure does not block any SAS Federation Server from receiving and processing transactions. SAS Federation Server URLs are listed, space delimited, in `$SAS_Activity_Resource`. If a SAS Federation Server fails, an asynchronous thread periodically tests to see whether the server has come back online. If the server has come back online, the engine automatically re-creates an associated connection pool and brings the SAS Federation Server back into the cluster. This architecture makes the full processing capacity of the server cluster available to all processes. It also maximizes the retention of processing

capacity in the event of a server failure. However, this configuration also requires calls to be made across machine boundaries.

Alternatively, each middle-tier engine can be collocated with one SAS Federation Server. This deployment choice has the advantage of reducing calls across a network. It also simplifies the process of scaling up by adding servers, in that it does not require the analysis of middle-tier versus server-tier workloads.

The following activity types use the JDBC Connection system resource:

- SAS Activity
- General I/O Activity

JDBC Support for Enhanced Password Encryption for System Resources

JDBC system resources for SAS Decision Services hold the user name and password information used to connect to the database server or the SAS Federation Server. By default, this password is encoded using the sas002 encoding and held in the resource definition. However, some deployments might require stronger standards of encryption, as well as a centralized location to store the user name and password. It is possible for system resources to use log-in information that is defined in the SAS Metadata Repository as credentials to connect to the database or the SAS Federation Server.

Every user or group that is defined in the SAS Metadata Repository can hold several log-in accounts, each containing an authentication domain name, a user ID, and a corresponding password. It is possible to configure the SAS Metadata Server to store these passwords, encrypted using the AES (SAS003) and AES/FIPS (SAS004) algorithm.

Note: The instructions for configuring the SAS Metadata Server are described in *SAS Intelligence Platform: System Administration Guide*. You must have SAS/SECURE installed to support AES or AES/FIPS encryption.

Both the SAS Decision Services engine server and the design server use system resources to connect to databases or the SAS Federation Server. The SAS Decision Services engine uses the SAS Trusted User identity to access secure data. The design server uses the identity of the actual caller to do the same.

The value in the user name entry in the system resource is used to determine the credentials to connect to the server. The system first attempts to match the value with a SAS Server definition in the SAS Metadata Repository. The following scenarios are dependent on whether a match is found:

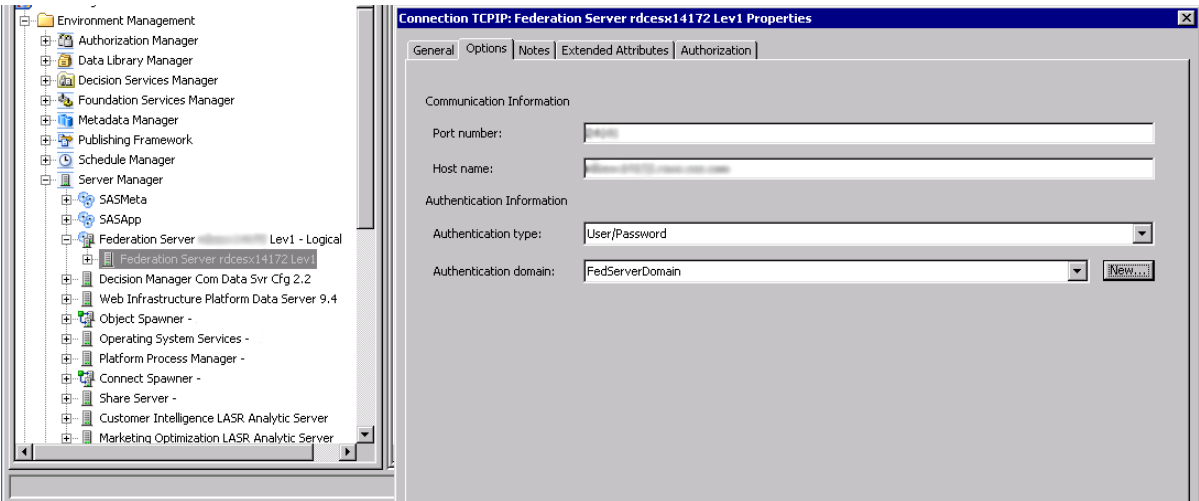
- If there is a match, the authentication domain that is associated with the server is used to select the log-in information that is available to the user. The log-in information is used to connect to the server that is referenced by the system resource. The password entry in the system resource is ignored.
- If the user name does not match any server name, the system searches the available log-in options of the user for one that has an authentication domain that matches the user name entry. The first such log-in information to match is used to connect to the server that is referenced by the system resource. The password entry in the system resource is ignored. If no match is found, the system reverts to using the user name and the password to connect to the server.

Note: If a match is found, the system uses the credentials that are associated with the first match. If the credentials are incorrect or invalid, the system does not try to use other possible matches.

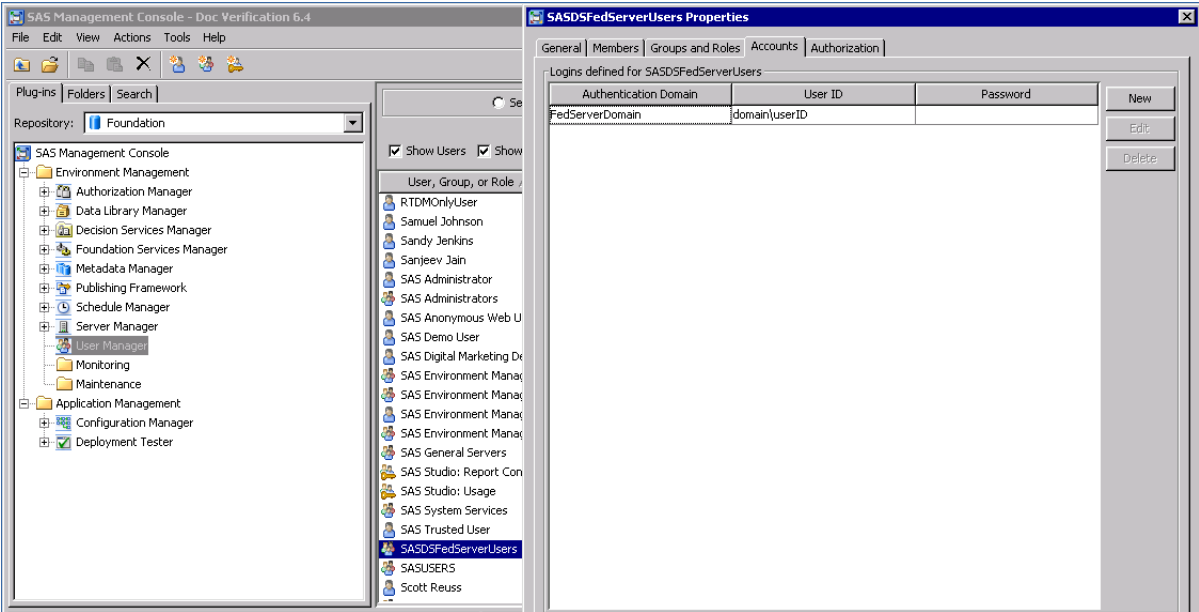
It is recommended that you create a group and assign all users who need access to the resource to it. The log-in information for the appropriate domain should be created for the group and not individual users. Separate groups can be created for the SAS Federation Server, databases, engine server usage, design server usage, and others, as required.

Here is an example of using the enhanced password encryption:

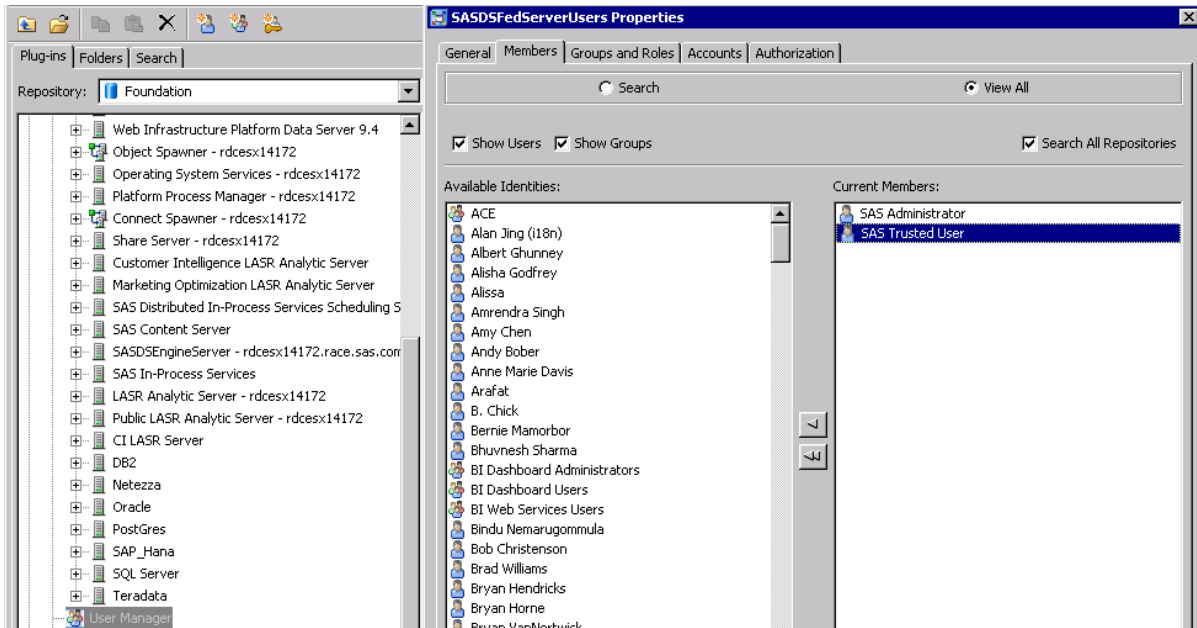
- 1 Create an authentication domain called FedServerDomain. Assign it to the connection of the server using the Server Manager plug-in in SAS Management Console.



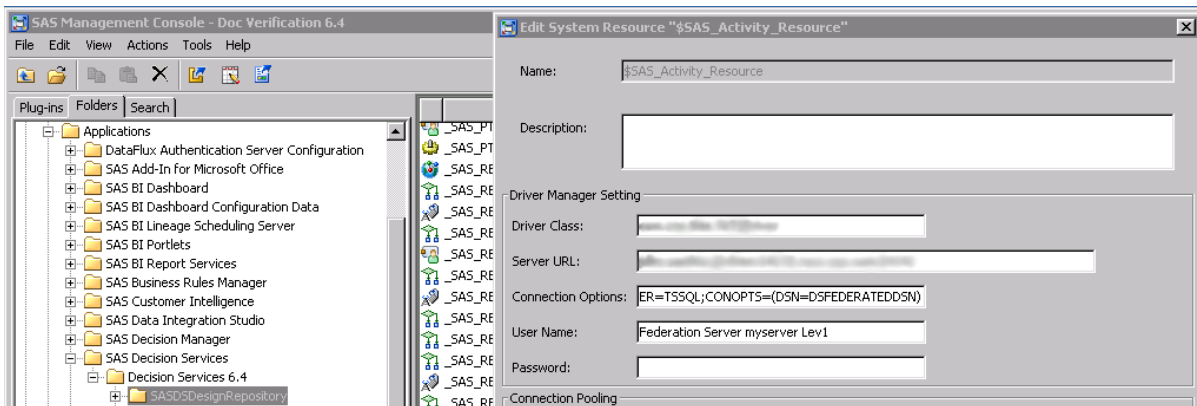
2 Create a group called SASDSFedServerUsers, using the User Manager plug-in in SAS Management Console.



3 Create log-in information for the group SASDSFedServerUsers, for the FedServerDomain authentication domain, with the appropriate user ID and password that are needed to connect to the SAS Federation Server. Assign the SAS Trusted User, the SAS Administrator, and other appropriate users to this group.



- 4 From the **Folder** tab, navigate to **System ► Applications ► SAS Decision Services ► Decision Services 6.4**, and edit the \$SAS_Activity_Resource system resource. Change the value in the **User Name** field to Federation Server *servername* Lev1. Clear the **Password** field.



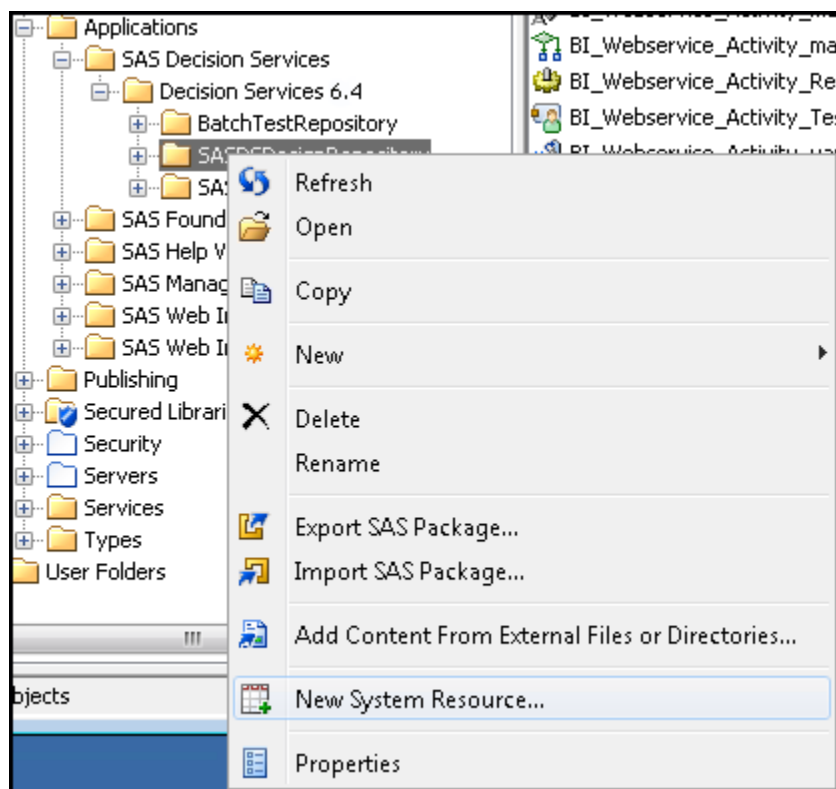
- 5 Restart the SAS Decision Services web applications. Verify that the SAS Decision Services engine and design servers can connect to the SAS Federation Server by viewing the diagnostics page.

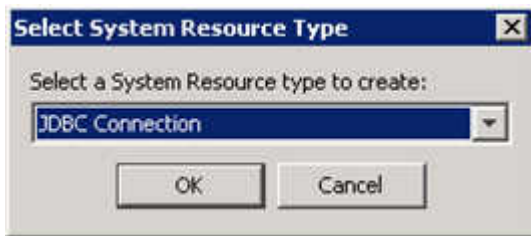
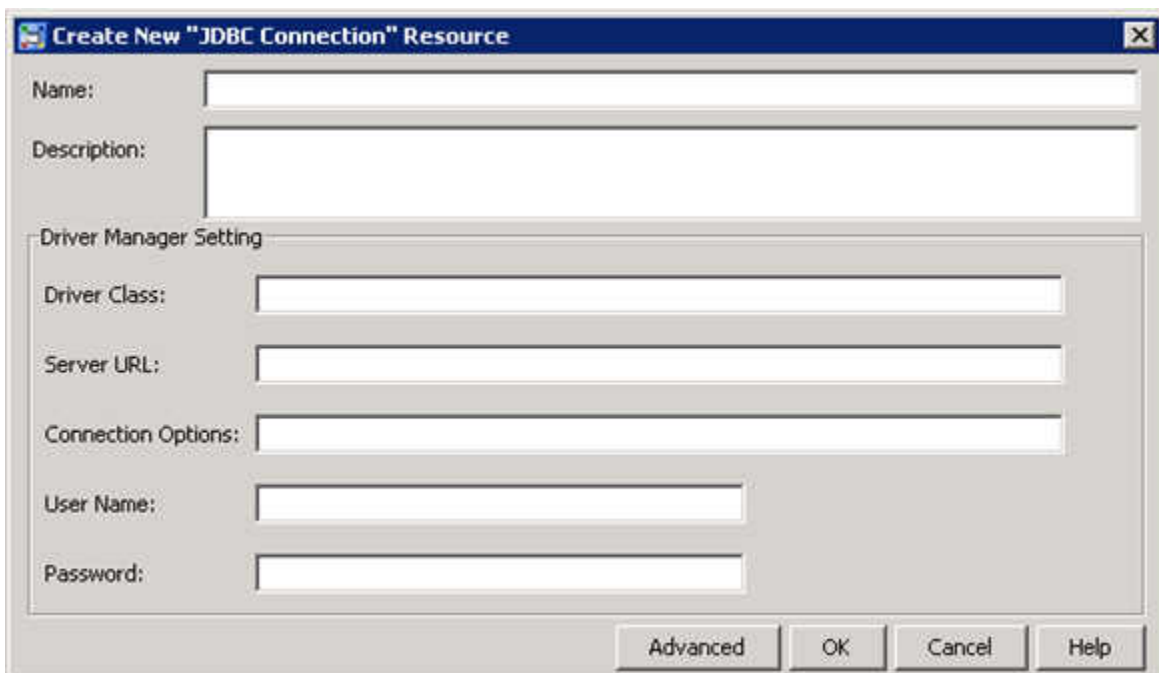
Use the same process to encrypt passwords for other JDBC connection resources that are connecting to other services. If your database server is not defined in the SAS Metadata Repository, you can create the domain when assigning it to the group, and enter the name of the domain in the **User Name** field of the system resource.

Specify a New System Resource as a JDBC Connection

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

- 1 Expand **System** ► **Applications** ► **SAS Decision Services** ► **Decision Services 6.4**.
- 2 Right-click a repository folder such as **SASDSDesignRepository**.
- 3 Select **New System Resource** from the drop-down menu.



4 Select JDBC Connection.**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. It 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.

Driver Class

specifies the Java class name of the database or SAS Federation Server driver. To create a resource for accessing database tables, use the class name of the driver that is provided by your database vendor. If you are unsure of what driver class name to use, see your system administrator.

Table 4.1 Supported Drivers for the Driver Class Field

Database	Class Name
DB2	com.ibm.db2.jcc.DB2Driver
Greenplum	org.postgresql.Driver
Netezza	org.netezza.Driver
Oracle	oracle.jdbc.driver.OracleDriver
PostgreSQL	org.postgresql.Driver
SAS Data Sets	com.sas.tks.TKTSDriver
SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver
Teradata	com.teradata.jdbc.TeraDriver

Server URL

is a database URL of the form jdbc:subprotocol:subname. See your system administrator for the URL that references your database installation. To create a system resource for executing DS2 activities, use the URL form jdbc:sastkts://host:port, where *host* and *port* reference your SAS Federation Server installation.

If this system resource is used for executing SAS activities, and if you have more than one SAS Federation Server in your environment (recommended), then enter a URL for each server, separating each URL with a space.

Table 4.2 Examples for the Server URL Field

Database	URL
Oracle	jdbc:oracle:thin:@//<server>:1521/ <database> For example: jdbc:oracle:thin:@// machine1.unx.sas.com:1521/sasds
SQL Server	jdbc:sqlserver://[machine1.na.sas.com]
Teradata	jdbc:teradata://machine1/
DB2	jdbc:db2://<server>:5000/<database> For example: jdbc:db2:// machine1.na.sas.com:50000/sasds
Greenplum	jdbc:postgresql://<server>:5432/ <database> For example: jdbc:postgresql:// machine1.unx.sas.com:5432/sasds
Netezza	jdbc:postgresql://<server>:5480/ <database> For example: jdbc:netezza:// machine1.unx.sas.com:5480/SASDS
PostgreSQL	jdbc:postgresql://<server>:5432/ <database> For example: jdbc:postgresql:// machine1.na.sas.com:5432/SASDS

Connection Options

(optional) use this field to create a resource for executing DS2 activities. The connection options should be in the form of DRIVER=TSSQL;CONOPTS=(DSN=Federation Server DSN).

For direct-to-database connections (general I/O), see the documentation for the specific database, to determine what options are available. With direct-to-database connections, the connection options are optional.

User Name

(optional) is used to connect to the database or SAS Federation Server that is specified in Server URL.

Password

(optional) is the password that is used to connect to the database or to the SAS Federation Server that is specified in Server URL, along with the user name.

(optional) Click **Advanced** to access connection and statement pool tuning controls. See the [“JDBC Performance Tuning” on page 176](#), for more information.

Specify a New System Resource as a Web Service Connection

The web service activity type is used to make direct requests to a web service as specified by the Web Service Connection system resource.

To specify a Web Service Connection as a system resource, follow steps 1–3 in [“Specify a New System Resource as a JDBC Connection” on page 72](#), and continue with these steps:

- 1 Select **Web Service**.
- 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.

Note: You must enter a valid URL for the WSDL. If the URL contains spaces and other disallowed characters, they must be encoded.

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), this field specifies your user password.

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

Specify a New System Resource as an HTTP Connection

You must specify the HTTP connection resource that the SAS Decision Services engine uses to communicate with servers that use HTTP (or HTTPS) as the transport protocol. The server capabilities are surfaced by specific activities that use this resource. The Name and URI fields are required.

To specify an HTTP Connection as a system resource, follow steps 1–3 in [“Specify a New System Resource as a JDBC Connection” on page 72](#), and continue with these steps:

- 1** Select **HTTP Connection**.
- 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. The name must be unique among the system resources.

Description

specifies additional information about the system resource. Description has a 200-character maximum length.

URI

a URI that follows the HTTP or HTTPS scheme. The URI references the server that this resource communicates with.

To configure the properties that are associated with this system resource click **Advanced**.

Library Resources

Overview

Library resources provide two distinct capabilities:

- To define alias names for database schemas
- To specify tables to cache in read-only memory

Note: Both of these features are optional and can be used together or separately.

(Optional) Define a Schema Alias

SAS Decision Services supports the optional use of aliases to reference database schemas.

For example, suppose your database has a schema called DDA, for direct-deposit accounts, and the SAS programs in your organization reference this schema by using a libref called ACCOUNTS. SAS Decision Services accesses data from your database directly, without going through SAS/ACCESS. Therefore, internally the SAS Decision

Services engine must use the actual schema name to access the tables within the schema.

For consistency with SAS, or to define user-friendly names, you might want to create an alias for DDA called ACCOUNTS by using a library resource.

Your SAS Decision Services repository can contain zero or more library resources. You must create a library resource for each schema alias that you want to define.

To specify a library resource, follow steps 1–3 in [“Specify a New System Resource as a JDBC Connection” on page 72](#), and continue with these steps:

- 1** Select **Library**.
- 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 library resource and the alias name to use. Host has a 60-character maximum length. Spaces are allowed.

Description

(optional) might describe the schema referenced by this library resource. Description has a 200-character maximum length.

Schema Name

the actual schema name defined to the database. Description has a 200-character maximum length.

Connection Resource

select the JDBC Connection system resource from the drop-down list, which references the database with the desired schema.

Databases

Overview

SAS Web Infrastructure Platform Data Server is included in your deployment for use as transactional storage by SAS Decision Services software. The server is based on PostgreSQL 9.x. The server is configured specifically to support SAS.

In a SAS Decision Services deployment, the server is configured to manage the DecisionServices database.

This database contains batch job execution and monitoring data that is generated by SAS Decision Services Monitor.

Connection Information for the JDBC Data Source

The database that is used by SAS Decision Services must be configured in SAS Web Application Server as a JDBC data source. The JDBC data source is configured with the JDBC driver and connection information for the selected database. These settings are provided to the SAS Deployment Wizard during installation and configuration.

The default database server for SAS Decision Services is the SAS Web Infrastructure Platform Data Server. The JDBC connection parameters for the server are provided in the following table:

Table 4.3 *JDBC Connection Parameters for SAS Web Infrastructure Platform Data Server*

Connection Parameter	Setting
JNDI name:	sas/jdbc/DecisionServices

Connection Parameter	Setting
JDBC URL:	jdbc:postgresl://serverName:port/ DecisionServices In the URL, substitute the server name and port number of the SAS Web Infrastructure Platform Data Server at your site. The default port is 9432.
JDBC driver class:	org.postgresql.Driver

These settings are configured during initial deployment. However, note the connection information so that you can supply it if you make changes later, such as moving the server to another host system.

Note: You must specify the user name and password values as required to access the data source.

These settings are represented in SAS Web Application Server in the *SAS-config-dir\Levn \Web\WebAppServer\SASServer7_1\conf\server.xml* file:

```
<Resource auth="Container" driverClassName="org.postgresql.Driver"
    factory="com.sas.vfabrictcsvr.atomikos.BeanFactory" maxPoolSize="100"
    minPoolSize="10" name="sas/jdbc/DecisionServices"
    password="{pw.sas.jdbc.DecisionServices}"
    testQuery="select 1 from monitor.dcsv_topology"
    type="com.atomikos.jdbc.nonxa.AtomikosNonXADataSourceBean"
    uniqueResourceName="sas/jdbc/DecisionServices"
    url="jdbc:postgresql://hostname.example.com:9432/DecisionServices"
    user="DecisionServices"/>
```

The postgresql.jar JAR file provides the org.postgresql.Driver class. SAS provides the JAR file in the *SASHOME\SASWebInfrastructureDataBaseJDBCDrivers \9.4\Driver* directory.

5

Decision Services Activities

Overview	84
SAS Activities	85
What Is a SAS Activity?	85
Creating a New SAS Activity	85
Out of the Box SAS Activity DS2 Package	88
Accessing Database Tables from a Custom SAS Activity or from a Business Rules Node	102
Web Service Activities	105
Invoking External Web Service Activities	105
Invoking SAS BI Web Services	106
SAS Customer Experience Real-Time Server Engine Integration Activity	107
General I/O Activities	109
Overview	109
Operations	110
Library Resources	116
Guidelines for Creating Activities	117
Date and Time Formats That Are Supported by SAS Decision Services	117
Boolean Values	117
General I/O Write and SAS Data Sets	118

Overview

SAS Decision Services provides a rich set of activities 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 business rules, and executing scoring models.

If your organization has a special processing need that is not covered by the provided activity set, new activities can be added. This is accomplished by developing custom SAS code and publishing it to the SAS Decision Services environment. The activity publishing step assembles metadata. Metadata is necessary in order for the activity to be recognized by a SAS Decision Services engine and to be rendered and tested in a client environment, such as SAS Customer Intelligence Studio or SAS Enterprise Decision Manager. The user interface that is used to publish activities is provided by the SAS solution, such as SAS Customer Intelligence, which in turn makes SAS Decision Services API calls in order to publish a new activity.

SAS Decision Services uses the following classifications of configurable activities:

- SAS activity
- web service activity
- general I/O activity

The SAS activity type is used to host score code and business rules. It is also used to extend SAS Decision Services functionality. A SAS activity consists of a SAS program and an activity XML document that describes the activity, the methods that are supported by that activity, and the system resources that are used by that activity.

DS2 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

What Is a SAS Activity?

SAS activities are powerful tools for expanding the functionality of SAS Decision Services. The code of a SAS activity corresponds to a DS2 package. A DS2 package is an object containing a set of associated methods that perform specific functions, similar to a class in an object-oriented programming language. The DS2 package is given the same name as the activity that it implements.

Creating a New SAS Activity

Overview

Activities are published using the solution that incorporates SAS Decision Services, such as SAS Real-Time Decision Manager. To create a new SAS activity, first create a DS2 package that contains the SAS code to be executed. If your activity is to be used with SAS Real-Time Decision Manager, your DS2 package must contain the method `execute()`. SAS Real-Time Decision Manager will call only the `execute()` method of an activity from a flow. However, you can include other DS2 methods and call them from your `execute()` method. Also, you must give your package the same name as your activity. Follow the instructions that came with your solution to publish your new activity. The solution sends the activity code and metadata to the SAS Decision Services Design Server, which stores it in the design repository folder within SAS Metadata Server. After the package has been completed and stored in the repository, you can create flows that include the SAS activity.

Efficiency Considerations

Consider efficiency first when developing SAS activity code. If your decision flows are required to provide an immediate response, avoid implementing long-running processes such as table joins, non-indexed searches, nested loops, or expensive database queries. Remember that a decision flow executes no faster than the cumulative speeds of the activities that it contains.

Create a DS2 Package

Create your DS2 package in an interactive SAS session. This method enables you to conduct immediate testing to be sure the code is correct. DS2 packages are created using PROC DS2. For more information about PROC DS2, see *SAS 9.4 DS2 Language Reference*.

- 1 Set the NOPROMPT option in your PROC DS2 statement to point to your design or test SAS Federation Server. This ensures that the version of SAS used to compile your DS2 activity matches the version that is used by SAS Decision Services at run time.

```
proc ds2 nolibs noprompt="driver=remts;server=your_Fed_Server;port=21032;
protocol=bridge;uid=user;pwd=password;
conopts=(DSN=Fed_Server_DSN) ";
  ds2_options sas;package my_pkg /overwrite=yes sas_encrypt=yes;
    method execute(vvarchar(32767) in_string, in_out vvarchar out_string);
      out_string=in_string;
    end;
  endpackage;
run;
quit;
```

This code creates a package that is called `my_pkg` that contains one method, `execute()`, and stores it in the database that is pointed to by `Fed_Server_DSN`. SAS activity methods must be coded as void functions in DS2. Output parameters must be marked with the `in_out` tag, which causes their values to be returned to the middle tier after method execution.

Note: SAS Decision Services does not support the use of `in_out` tagged parameters for input. They are used strictly for output only. Also, always ensure that all input arguments precede the output arguments within your DS2 method signatures.

To force a package to always execute in SAS missing mode, use `ds2_options sas;` as the first statement, before the `PACKAGE` statement. When you omit this option, your package uses ANSI missing mode by default. SAS missing mode is recommended to achieve the highest compatibility between DS2 and DATA step.

Your custom activity code might include more than one DS2 package. The methods of the last package in your DS2 program are the only methods that are visible to decision flows. If you are using SAS Real-Time Decision Manager, then the `execute()` method of the last package is the only method that can be called by the

decision flows. The arguments for these methods must use only the Decision Services data types. Otherwise, an error is returned during the activity publishing step.

You can test your package in your interactive SAS session by using a DS2 TABLE statement:

```
proc ds2 nolibs conn="driver=remts;server=your_Fed_Server;port=21032;
protocol=bridge;uid=user;pwd=password;
conopts=(DSN=Fed_Server_DSN) ";
  table _null_;
    method init();
      dcl package my_pkg echo();
      dcl varchar(32767) out_string;
      echo.echo_string('String to echo', out_string);
      put out_string=;
    end;
  endtable;
run;
quit;
```

- 2** When you publish a new or modified SAS activity in the design environment, the activity is immediately made available for inserting into flows and for testing. After changing and republishing an existing SAS activity in a test or production environment, or after importing a modified SAS activity into a test or production environment, the engine picks up the new activity code the next time a flow is activated or deactivated. Until that time, any prior version of the activity continues to be used.

Create SAS Activity XML

SAS Decision Services client applications, such as SAS Customer Intelligence, provide an interface for entering activity metadata. For more information, see your client application's documentation.

- 1** Using the client application, create a new activity.
- 2** Give the client application the location of the .sas file containing the DS2 source code for your activity.
- 3** The activity name matches the DS2 package name that was created earlier.

- 4 Enter a description that includes you as the owner and that describes the purpose of the activity. This is good practice that enables you to better manage your files.
- 5 Enter methods that match each method in your DS2 package. The order of the parameters in the method is important and must match the order of parameters in the DS2 package method.

Note: SAS Customer Intelligence recognizes only a single method per activity called "execute."

Data Type Mappings

The following table lists the SAS Decision Services data types and the corresponding DS2 data types. When you create a flow or event, you work with the data types in the left column. When you write DS2 code, you use the data types in the right column.

Note: DateTime fields contain SAS datetime values. Datetime values are the number of seconds since January 1, 1960.

Table 5.1 Data Types

SAS Decision Services Data Type	DS2 Data Type
String	Varchar
Int	Bigint
Float	Double
Boolean	Integer
DateTime	Double

Out of the Box SAS Activity DS2 Package

Overview

Several DS2 packages are provided out of the box, in the following location `<Lev Config Dir>\Applications`

`\SASDecisionServicesServerConfig6.4\SASCode`. Some of those SAS files are required by the SAS Decision Services run time, some are utilities to help you build your own SAS activities, and some are sample SAS activities.

Note: Do not modify these DS2 packages. Do not call methods that are marked as internal-use-only in the source code comments.

Utility Packages

Note: To add logging to your custom DS2 packages, see “DS2 Logger Package Methods, Operators, and Statements” in *SAS DS2 Language Reference*.

tap_hash

This package is a simple extension of the DS2 hash object. Each new package does not need to declare its own hash extension package; this one is provided for everyone’s use. Reference this package by specifying the database catalog name, a period, and then `tap_hash`.

tap_array

SAS Decision Services array objects are passed to a DS2 method as an encoded string (varchar) parameter. Use the `tap_array` package to decode the string. Empty array objects can also be created and populated by your custom SAS activity code. This package provides an `encode()` method that can be called to create an encoded string version of the current array. This is the array that is to be returned to the SAS Decision Services engine.

Here are the available methods:

- `tap_array();` - Constructs an empty array.

Note: `set_type` must be called immediately after using this constructor.

- `tap_array(varchar input_array);` - Constructs an array that is initialized using the encoded `input_array` string.
- `set_type(varchar type);` - Sets the type of array. Choose one of the following types: `STRING`, `INT`, `FLOAT`, `BOOLEAN`, or `DATETIME`. This method is not case sensitive.
- `type()` returns varchar; - Returns the type of array. Choose one of the following: `STRING`, `INT`, `FLOAT`, `BOOLEAN`, or `DATETIME`.

- `encode()` returns `varchar`; - Encodes this array into a string for return to the SAS Decision Services engine.
- `add(varchar element)`; - Appends the specified element to the end of this array.
- `add(int element)`; - Appends the specified element to the end of this array.
- `add(double element)`; - Appends the specified element to the end of this array.
- `add(int index, varchar element)`; - Inserts the specified element at the specified position in this array.
- `add(int index, int element)`; - Inserts the specified element at the specified position in this array.
- `add(int index, double element)`; - Inserts the specified element at the specified position in this array.
- `addAll(package tap_array in_array)`; - Appends all of the elements in the specified array to the end of this array.
- `clear()`; - Removes all of the elements from this array.
- `set_null()`; - Sets this array to null.
- `getString(int index)` returns `varchar`; - Returns the element at the specified position in this array.
- `getInt(int index)` returns `int`; - Returns the element at the specified position in this array.
- `getDateTime(int index)` returns `double`; - Returns the element at the specified position in this array.
- `getBoolean(int index)` returns `int`; - Returns the element at the specified position in this array.
- `getFloat(int index)` returns `double`; - Returns the element at the specified position in this array.
- `isEmpty()` returns `int`; - Returns 1 (true) if this array contains no elements, 0 (false) otherwise.

- `delete(int index);` - Deletes the element at the specified position in this array.
- `setString(int index, varchar element);` - Replaces the element at the specified position in this array with the specified element.
- `setInt(int index, int element);` - Replaces the element at the specified position in this array with the specified element.
- `setFloat(int index, double element);` - Replaces the element at the specified position in this array with the specified element.
- `setDateTime(int index, double element);` - Replaces the element at the specified position in this array with the specified element.
- `setBoolean(int index, int element);` - Replaces the element at the specified position in this array with the specified element.
- `size()` returns `int`; - Returns the number of elements in this array.

tap_table

Here are the available methods:

- `tap_table();` - Creates an empty table.
- `tap_table(varchar input_table);` - Creates a table that is initialized with the input table string.
- `encode()` returns `varchar`; - Encodes the table into a string that can be passed back to the SAS Decision Services engine.
- `add_column(varchar name, varchar type);` - Adds a column of the given type to the table.
- `add_row();` - Adds a new row to the table, all values are set to null.
- `add_row(int rows);` - Adds the specified number of rows to the table, all values are set to null.
- `column_count()` returns `int`; - Returns the number of columns in the table.
- `row_count()` returns `int`; - Returns the number of rows in the table.

- `column_name(int index)` returns varchar; - Returns the name of the column at the given ordinal.
- `column_type(int index)` returns varchar; - Returns the type of the column at the given ordinal.
- `column_type(varchar name)` returns varchar; - Returns the type for the given column.
- `decodeJSON()` - Initializes a `tap_table` instance using the given JSON string.

Note: For more information about the expected dialect of the JSON text, see [“Using the tap_table Package decodeJSON and encodeJSON Methods” on page 99.](#)

- `delete_column(varchar name);` - Removes the given column from the table.
- `delete_row();` - Removes the given row from the table.
- `encodeJSON()` - Returns a nvarchar string, where the string is a JSON representation of the `tap_table` instance.

Note: If the format of your data does not agree with the default values, see [“Using the tap_table Package decodeJSON and encodeJSON Methods” on page 99.](#)

- `getString(varchar col_name)` returns varchar; - Retrieves the string value from the given column at the current row.
- `getInt(varchar col_name)` returns int; - Retrieves the int value from the given column at the current row.
- `getBoolean(varchar col_name)` returns int; - Retrieves the Boolean value from the given column at the current row.
- `getFloat(varchar col_name)` returns double; - Retrieves the float value from the given column at the current row.
- `getDateTime(varchar col_name)` returns double; - Retrieves the datetime value from the given column at the current row.

- `setString(varchar col_name, varchar element);` - Sets the value of the given column, at the current row, to the given string value.
- `setInt(varchar col_name, int element);` - Sets the value of the given column, at the current row, to the given int value.
- `setFloat(varchar col_name, double element);` - Sets the value of the given column, at the current row, to the given float value.
- `setDateTime(varchar col_name, double element);` - Sets the value of the given column, at the current row, to the given datetime value.
- `setBoolean(varchar col_name, int element);` - Sets the value of the given column, at the current row, to the given Boolean value.
- `set_null();` - Sets the table to null.

DS2 RESTCallout

The DS2 RESTCallout package provides a means to execute HTTP POST requests from DS2. It is a helper package that is a front end to the HTTP DS2 package. More specific requests might require the direct usage of the DS2 HTTP package, when the use of specific DS2 HTTP package features is desired.

Here are the available methods:

tap_RESTCallout

The constructor method creates a DS2 RESTCallout package instance. Its member variables include a `varchar(2048)` URL attribute, as well as HTTP and `tap_logger` package instances.

createPost

The `createPost` method creates an HTTP request, setting its URL and allowing you to provide other parameters, such as a timeout and the `contentType`, and to accept HTTP headers. A call to `createPost` can be followed by many `executePost` calls. Only one URL can be active per RESTCallout instance. Calling `createPost` replaces the URL of a previous `createPost` call. Here are the signatures for `createPost`:

```
createPost( url, timeout, userid, password, rc );
createPost( url, contentType, accept, timeout, userid, password, rc );
```

Here are the arguments for createPost:

- `nvarchar(16384)` URL - The web server's URL.
- `nvarchar(16384)` contentType - The type of payload. The default is `application/json; charset=utf-8`.
- `nvarchar(16384)` accept - The type of acceptable response. The default is `application/json; charset=utf-8`.
- `int` timeout - The timeout in milliseconds of each underlying tkhttp client request that is executed. This is not an overall cumulative timeout for the POST.
- `nvarchar(256)` userid - This is ignored.
- `nvarchar(256)` password - This is ignored.
- `in_out int` rc - The input rc variable is updated with the return code.

executePost

The executePost method executes an HTTP POST request and updates its output arguments. If the URL and timeout are given, createPost is called with those given arguments. If the signature without the URL is used, the previously created POST request is used. Here are the signatures for executePost:

```
executePost( payload, hstat, rc, responseBody );
executePost( payload, hstat, rc, responseBody, url );
executePost( payload, hstat, rc, responseBody, url, timeout,
            timeout, userid, password );
```

Here are the arguments for executePost:

- `nvarchar(67108864)` payload - The given body of the POST request.
- `in_out int` hstat - updated with the HTTP status code of POST request execution.
- `in_out int` rc - The method return code.
- `in_out nvarchar` responseBody - The response from the request.
- `nvarchar(16384)` url - The web server's URL.

- `int timeout` - The timeout in milliseconds.
- `nvarchar(256) userid` - This is ignored.
- `nvarchar(256) password` - This is ignored.

tap_datetime

The package `tap_datetime` wraps native SAS functions, passing a datetime or date number, as needed, to these functions.

- `tap_datetime()` - Creates a new instance with the date set to January 1, 1960.

Note: No local offset can be applied to any numbers that represent SAS dates or datetimes and that are passed to `tap_datetime()`. Therefore, those numbers have no local offset applied when they are returned from `tap_datetime()` methods.
- `tap_datetime(package tap_datetime)` - Creates a copy of the given `tap_datetime`.
- `tap_datetime(double sasDatetime)` - Creates a new instance that is based on the given SAS datetime (seconds since January 1, 1960).
- `tap_datetime(varchar stringRepresentation)` - Creates a new instance from the given string representation. The following formats are supported:
 - `'DDMMMYYYY'`, for example: `'15Mar2007'`
 - `'DDMMMYYYY:HH:MM:SS'`, for example: `'15Mar2007:15:30:45'`
 - `'YYYY-MM-DDTHH:MM:SS'`, for example: `'2007-03-15T15:30:45'`
- `varchar toString()` - Returns a string representation of this instance in the form of `'DDMMMYYYY:HH:MM:SS'`.
- `double toSASDatetime()` - Returns the SAS datetime (seconds since January 1, 1960) corresponding to this instance.
- `double toSASDate()` - Returns the SAS date (days since January 1, 1960) corresponding to this instance.
- `fromSASDatetime(double sasDatetime)` - Sets time for this instance based on the given SAS datetime (seconds since January 1, 1960).

- `fromSASDate(double sasDate)` - Sets time for this instance based on the given SAS date (days since January 1, 1960).
- Package `tap_datetime` supports the following native SAS functions, but unlike their SAS equivalents, these functions take no input arguments. Instead, the values that are returned depend on the date and time that the `tap_datetime` instance represents.

For example, suppose you have an instance of package `tap_datetime` called “vacation” that is set to the value “12Apr2013”. Then a call to `vacation.year()` would return the value 2013.

For complete descriptions of the native SAS functions, see *SAS DS2 Language Reference*.

The advantage to using the `tap_*` packages is that they correctly call the equivalent SAS methods. Some of the following SAS methods require a SAS date, and some require a SAS datetime.

- ☐ `int year()`
- ☐ `int month()`
- ☐ `int day()`
- ☐ `int hour()`
- ☐ `int minute()`
- ☐ `int second()`
- ☐ `int weekday()`
- ☐ `int qtr()`
- ☐ `double timepart()`
- ☐ `double datepart()`

`tap_datetime_utilities`

The package `tap_datetime_utilities` contains logically static functions that construct `tap_datetime` instances, or that operate on more than one `tap_datetime` instance.

- `tap_datetime_utilities()` - Constructs a new instance.

- `package tap_datetime datetime()`, `package tap_datetime today()`, `package tap_datetime date()` - These methods are equivalent. They return a `tap_datetime` instance with the time set to current time.
- `package tap_datetime dhms(package tap_datetime dt, int hours, int minutes, int seconds)` - Returns a new `tap_datetime` instance that is equal to the given `tap_datetime` argument, with hours, minutes, and seconds reset to the given values. This is equivalent to the SAS function `dhms()`.
- `package tap_datetime mdy(int month, int day, int year)` - Returns a `tap_datetime` instance that is constructed from the given values. This is equivalent to the SAS function `mdy()`.
- `package tap_datetime yyq(int year, int quarter)` - Returns a `tap_datetime` instance that is constructed from the given values. This is equivalent to the SAS function `yyq()`.
- `int datdif(package tap_datetime dt1, package tap_datetime dt2, varchar basis)` - Returns the difference between two `tap_datetimes` in days. This is the equivalent to the SAS function `datdif()`.
- `package tap_datetime SASDatetimeToDatetime(double sasDatetime)` - Returns a `tap_datetime` instance that is constructed from the given SAS datetime (seconds since January 1, 1960).
- `package tap_datetime SASDateToDatetime(double sasDate)` - Returns a `tap_datetime` instance that is constructed from the given SAS date (days since January 1, 1960).

Sample Package

`sas_activity_tests`

This is a sample package that can be used for validation and testing. Here are the available methods:

`echo_string`

Signature - `(varchar(32767) in_string, in_out varchar out_string)`

Description - Echoes the input string to the output string.

echo_int

Signature - (int in_int, in_out int out_int)

Description - Echoes the input int to the output int.

echo_float

Signature -(double in_float, in_out double out_float)

Description - Echoes the input float to the output float.

echo_boolean

Signature - (int in_boolean, in_out int out_boolean)

Description - Echoes the input Boolean to the output Boolean.

echo_datetime

Signature - (double in_datetime, in_out double out_datetime)

Description - Echoes the input datetime to the output datetime.

echo_scalars

Signature - (varchar(32767) in_string, int in_int, double in_float, int in_boolean, double in_datetime, in_out varchar out_string, in_out int out_int, in_out double out_float, in_out int out_boolean, in_out double out_datetime)

Description - Echoes the input values to the output values.

echo_array

Signature - (varchar(32767) in_array, in_out varchar out_array)

Description - Echoes the input array to the output array.

echo_table

Signature - (varchar(32767) in_table, in_out varchar out_table)

Description - Echoes the input table to the output table.

variable_test

Signature - (varchar(32767) in_string, bigint in_int, double in_float, int in_boolean, double in_datetime, varchar(32767) in_array, varchar(32767) in_table, in_out varchar out_string, in_out bigint out_int, in_out double out_float,

in_out int out_boolean, in_out double out_datetime, in_out varchar out_array,
in_out varchar out_table)

Description - Edits each of the input values and sets them in the output values.

- out_string - The result of reversing in_string. For example, "abc" becomes "cba."
- out_int - The result of in_int + 2.
- out_float - The result of in_float + 1.11.
- out_boolean - The negation of in_boolean - true = false and false = true.
- out_datetime - The result of out_datetime + 1 day.
- out_array - The reverse array order of in_array - String1, String2, String3 becomes String3, String2, String1.
- out_table - The input table with the row order reversed, 100 added to each column of type int, 222.222 added to each column of type float, 6 days added to each column of type datetime, the string reverse for each column of type string, and the negation for each column of type Boolean.

Using the tap_table Package decodeJSON and encodeJSON Methods

The decodeJSON method first clears the tap_table data, and then initializes the tap_table instance with the given JSON text input argument. The decodeJSON method returns a return code type of int. It has one signature, which has one nvarchar input argument. Here is the method signature:

```
method decodeJSON( nvarchar(67108864) input_table ) returns int;
```

The decodeJSON method's nvarchar string input argument is expected to contain JSON text, conforming to the following JSON dialect:

```
[ { "metadata":
    [ { "<var_name_1>": "<type>" },
      { "<var_name_2>": "<type>" },
      ...
      { "<var_name_n>": "<type>" } ] },
  { "data":
    [ [ <obs 1 data> ],
```

```
[ <obs 2 data> ],
...
[ <obs n data> ] ] }
```

Note: White space is ignored.

The following sample table illustrates the dialect:

```
[ { "metadata":
  [ { "myInt": "integer" },
    { "myFloat": "decimal" },
    { "myString": "string" },
    { "myBoolean": "boolean" },
    { "mydatetime": "dateTime" } ] },
  { "data":
    [ [ 1, 1.1, "first", false, "1960-01-01T00:00:01Z" ],
      [ 2, 2.2, "second", false, "1960-01-01T00:00:01Z" ],
      [ 3, null, "third", false, "1960-01-01T00:00:01Z" ]
    ] } ]
```

A null table is encoded as simply "null." A table with no rows has an empty data array.

The encodeJSON method returns a nvarchar string consisting of the tap_table instance formatted into JSON text. The encodeJSON method has two signatures, one without any input arguments, and the other with three:

```
encodeJSON( int dblWidth, int dblPrecisn, varchar(128)
            dblFmtSpecifier ) returns nvarchar;
```

The signature without args calls the other using 0, 15, and "BESTFIT" as its args. Here is an example:

```
method encodeJSON( int dblWidth, int dblPrecisn,
                  varchar(128) dblFmtSpecifier ) returns nvarchar;
```

The implementation of the encodeJSON method signature without input arguments is as follows:

```
method encodeJSON() returns nvarchar;
  return( encodeJSON( 0, 15, 'BESTFIT' ) );
end;
```

The encodeJSON method arguments are as follows:

- **int dblWidth** - Specifies the width for all numeric values that are written. The value represents the maximum width for SAS formatting, and the minimum width for printf formatting. The values are left-justified.

Note: Additional details for SAS and printf formatting are provided below.

- `int dblPrecisn` - Specifies the precision for all numeric values that are written. Precision indicates the number of digits that appear after the radix character.
- `varchar(128) dblFmtSpecifier` - Specifies one of the following printf or SAS format type specifiers. The BESTFIT specifier is the default.

Printf type specifiers:

BESTFIT (default)

This is the default specifier. Format the number using a decimal notation style in the form of `[-]dddd.ddd`, or scientific notation in the form `[-]d.ddde±dd`. The formatting style of depends on the value.

BESTFITBIG

This is the same as BESTFIT, except that the uppercase “E” is used in place of the lowercase “e”, if scientific notation style is used.

DECIMAL

Format the number using decimal notation in the form of `[-]dddd.dd`, using the precision to determine the number of digits after the radix. The radix character does not appear if there are no digits to display after it or if the precision is set to 0.

FRACTION

Format the fractional part of the double in the form of `[-]0.dddd`. The digit before the radix character is always 0.

INTEGER

Format the integer part of the double in the form of `[-]dddd`. The fractional part of the value is ignored, and no radix character is added to the result.

SNOTE

Format the number as a scientific notation in the form of `[-]d.ddde±dd`, using the lowercase 'e' to precede the exponent.

SNOTEBIG

Format the number as a scientific notation in the form of `[-]d.dddE±dd`, using the uppercase 'E' to precede the exponent.

SAS format type specifiers:

SASBEST

Conforms to the SAS BESTw. format rules. The value is formatted within the specified width. Decimal notation is produced if possible. Otherwise, scientific notation is produced in the form of [-]ddd.dddE[-]dd. Trailing zeros after the radix character are suppressed.

SASEW

Conforms to the SAS Ew. format rules. The values are formatted within the specified width. Scientific notation is always produced in the form of [-]ddd.dddE±dd.

SASEWD

The value is formatted with the given width precision. Scientific notation is always produced in the form of [-]ddd.dddE±.dd.

SASWD

The value is formatted with the given width precision. Scientific notation is always produced in the form of [-]ddd.dddE±.dd.

Accessing Database Tables from a Custom SAS Activity or from a Business Rules Node

The preferred vehicle for accessing a database is the General I/O activity. However, there might be times when it is advantageous for custom SAS code to do so.

To enable SAS activity or business rule code to read from, or write to, a database, you must first create a federated DSN. Federated DSNs contain a list of standard DSNs, enabling access to more than one data source. By referencing the federated DSN in your connection string, you gain access to all of the catalogs and schemas that are referenced by the contained DSNs. For more information about federated DSNs, see *SAS Federation Server 4.1: Administrator's Guide*.

Note: The default DSN in a federated DSN is the first DSN that is added. When you add a federated DSN through the command line utility, this is the first DSN on the list of DSNs. When adding a federated DSN through the UI, add only the default DSN first. Then, you can edit the newly created federated DSN and add any desired additional

DSNs. The default DSN is where DS2 packages are stored. The additional DSNs are used for data access from those DS2 programs.

Because DS2 packages are stored in SAS data sets, your federated DSN must include `BASE_DSN` as well as any additional DSNs that reference the database catalogs, schemas, and tables that you want to access.

To create a federated DSN, connect to SAS Federation Server Manager and log on to your federation server definition with a user ID that has administrative privileges, and follow these steps:

- 1** With the Federation Server definition selected, click the **Data Source Names** tab.
- 2** From the drop-down list, select **New Federated Data Source Name**.
- 3** Enter the name and description for the federated DSN, and click **Next**.
- 4** From the drop-down list, select **Add Data Source Names**.
- 5** Select the DSNs that you want to connect to with this federated DSN, and click **OK**.
- 6** When you return to the Members screen, click **Next**.
- 7** It is recommended that you keep the default security setting, and click **Next**.
- 8** When you have reviewed the information about the Summary screen, click **Finish**.

You can test your federated DSN by modifying the following SAS program:

```
proc ds2 Conn="driver=remts;server=your_server;port=your_port;protocol=bridge;
  uid=admin_userid;pwd=admin_password;conopts=(DSN=your_federated_dsn)";table _null_

  method run();
    set AN_EXISTING_DATABASE_CATALOG.SCHEMA.TABLE;
    put a_column= another_column=;
  end;

endtable;

run;
quit;
```

Custom SAS activities are implemented as DS2 packages. To read from a table from within a DS2 method, you must use either the DS2 hash package or the DS2 SQLStmt package. To write to a table from within a DS2 method, use the SQLStmt package. The hash package can be used only for reading. To read using a hash object, use the `dataset()` method of the DS2 hash object. This method takes an SQL SELECT statement as an argument and populates the hash object with the corresponding result set.

```
method compute();
  dcl package hash h();
  dcl package hiter hi(h);
  dcl int rc;

  h.definekey('clientid');
  h.definedata('hhid');
  h.definedata('income');
  h.dataset(
    '{select clientid, hhid, income from DSORA.MAFUNC.CUSTOMER1;}');
  h.definedone();

  rc = hi.first();
  do while(rc = 0);
    ...do something with the data...
    rc = hi.next();
  end;

end;
```

The SQLStmt package supports SQL syntax similar to that used in JDBC parameterized prepared statements. It also provides control over SQL statement lifetime, enabling more efficient code to be written. The following example illustrates writing five records to a database table called “testdata”:

```
dcl package sqlstmt s('insert into testdata (x, y, z) values (?, ?, ?)', [x y z]);

do i = 1 to 5;
  x = i;
  y = i*1.1;
  z = i*10.01;
  s.execute();
end;
```

If you want to access databases from custom DS2 code, the database must first be set up within the SAS Federation Server. In the set-up process, a data service is created.

Within that data service, there is a catalog name that is used for the service. That is the catalog name that should be used when referencing the database from within the custom DS2 code. The schema name depends on the database. However, it is usually the database schema name. For data sets, it is the schema name that is used when setting up the service in the SAS Federation Server. The table name is the actual name of the table in the database.

For more information, see *SAS 9.4 DS2 Language Reference*.

Web Service Activities

Invoking External Web Service Activities

SAS Decision Services functionality can be extended by adding new web service activities. A web service activity can invoke an external web service that requests information to be used downstream in the decision flow. For example, suppose an organization has an inventory system with a web service interface. It is possible to create a web service activity 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 maps leaf-level elements of the XML, for the request and response payloads, to SAS Decision Services process variables of the following data types:

- BOOLEAN
- INT
- FLOAT
- DATETIME
- STRING
- ARRAY OF BOOLEAN
- ARRAY OF INT
- ARRAY OF FLOAT

- ARRAY OF DATETIME
- ARRAY OF STRING

Web service activity supports only transport-level security using SSL (HTTPS).

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. For more information about the Web Service Connection system resource, see [“Specify a New System Resource as a Web Service Connection” on page 76](#).

Invoking SAS BI Web Services

SAS BI web services executes as SAS stored processes in the SAS server tier. The use of BI Web Services in real-time applications is not recommended, because of their slow execution speed. Neither sub-second latencies nor high transaction volumes can be supported when SAS BI web services are used. Unless you need to run a procedure, use a DS2-based SAS activity instead. SAS BI Web Service activity supports an extended set of data types. The standard web service activity supports the types that are listed above. The BI web service activity supports the following input and output parameter types:

Note: BI web service activities return only string arrays. They cannot return float arrays.

Table 5.2 Input Parameter Types

Stored Process Type	SAS Decision Services Type
Numeric (integer)	Int
Numeric (double)	Float
Text	String
Numeric (integer) with name ending in _b	Boolean

Numeric (integer) with name ending in _d	DateTime
Text with name ending in _a	String Array
Text with name ending in _t	Table

Table 5.3 *Output Parameter Types*

Stored Process Type	SAS Decision Services Type
Integer	Int
Double	Float
String	String
Integer with name ending in _b	Boolean
Integer with name ending in _d	DateTime
String with name ending in _a	String Array
String with name ending in _t	Table

Tables and arrays are passed in and out of the stored process as encoded strings. An autocall macro, called `scencode`, is provided to encode these objects.

BI web service activity supports only transport-level security using SSL (HTTPS).

SAS Customer Experience Real-Time Server Engine Integration Activity

SAS Customer Experience Real-Time Server Engine is a third-party product that collects information from customers who visit a website. It makes this information available as an XML-encoded document that can be retrieved from the SAS Customer Experience Real-Time Server Engine using an HTTP transport.

SAS Decision Services integrates with the SAS Customer Experience Real-Time Server Engine by providing an activity type called the XMLHttpActivity, as well as a resource type called the HttpResource.

As with other activity and resource pairs, there is a separation of workload. HttpResource is responsible for defining the location of the SAS Customer Experience Real-Time Server Engine, as well as how to reach it and system parameters that provide the most efficient data exchange and throughput. These values usually vary for different installations and deployments. XMLHttpActivity defines the data that is sent to, and retrieved from, the service. It provides the data to the decision flow as an activity method with input and output parameters.

SAS Decision Services provides an editor to create or edit resources of the type HttpResource. The activities of the type XMLHttpActivity are created by the client application using SAS Decision Services, in this case the Customer Intelligence solution SAS Real-Time Decision Manager.

The editor for HttpResource is made available through the SAS Management Console. Like other SAS Decision Services resource editors, it can be invoked by navigating to **System/Applications/SAS Decision Services/Decision Services 6.4**, right-clicking a SAS Decision Services repository folder, and selecting **New System Resource**. Alternatively, you can access the editor by right-clicking on a specific resource of this type and selecting **Edit System Resource**.

The editor allows the user to enter the name, description, and the URL of the SAS Customer Experience Real-Time Server Engine. It also provides a number of properties that can be used to tune the underlying software to create or manage the connections, in order to maximize performance. The software internally uses Apache Commons HTTP Client 4.0.1. The properties available for changing are the configuration parameters for the HTTP Client software and are described here: http://hc.apache.org/httpclient-3.x/preference-api.html#Supported_parameters. When a new resource is created, default values for most parameters are already set. It is recommended to start with these parameter values and then change them as part of a performance tuning exercise after measurement.

Note: Changes made by the editor do not immediately take effect in the engine. In most cases, a synchronize call has to be made to the engine.

The screenshot shows a window titled "Edit System Resource 'DSCXA'". It contains the following fields:

- Name:** DSCXA
- Description:** (empty)
- URI:** http://racesx06137.demo.sas.com:91

Below these fields is a "Properties" section containing a table with two columns: "Name" and "Value".

Name	Value
http.protocol.version	HTTP/1.1
http.protocol.expect-continue	false
http.protocol.cookie-policy	ignoreCookies
http.socket.timeout	1000
http.tcp.nodelay	true
http.connection.timeout	1000
http.connection.stalecheck	false
http.connection-manager.max-per-host	1000
http.connection-manager.max-total	1000
http.connection-manager.timeout	1000
http.connection-manager.class	org.apache.commons.httpclient.MultiThreadedHttp...

General I/O Activities

Overview

SAS Decision Services is shipped with a General I/O activity that can read or write to any supported database table or SAS data set. A General I/O activity uses a JDBC Connection resource. This resource specifies which database the activity uses. At least one JDBC Connection resource was configured when your system was installed.

Note: SAS data sets exhibit file-level locking. If multiple threads of execution attempt to simultaneously read from or write to a SAS data set, deadlocks can occur. Therefore, the use of a relational database management system is highly recommended for real-time (non-batch) processing.

CAUTION! SAS Decision Services interprets DateTime values that are stored in a database or in SAS data sets, as being in the Greenwich Mean Time (GMT) time zone. A default time zone (GMT) is associated with values that are read from a database because time zone information is not persisted with the data. As a best practice, always store DateTime values using GMT time zone. This practice enables compatibility with SAS Decision Services and removes ambiguity.

Operations

Read

Method name: SCReadTable.

Properties

G_IO_WHERE_Clause - WHERE clause. The WHERE clause property is a static string that is set on the General I/O Activity instance when it is inserted into a flow.

A WHERE clause is a SAS Decision Services (not SQL) Boolean expression.

Logical (AND, OR, NOT), relational (EQ, NE, GT, GE, LT, LE), and arithmetic (+, -, /, *) operators can be used. Here is an example: CustomerInfo.Income GT 50000.0.

As in a DATA step, a . (period) denotes a missing value.

Note: To ignore trailing blanks, use the operators

SQL_EQ,SQL_NE,SQL_GT,SQL_GE,SQL_LT,SQL_LE. These operators are valid only for strings.

Process parameters can be referenced as *:{Process parameter name}*. Here is an example: CustomerInfo.LastName EQ :PV_CustomerLastName

Note: '=' and '!=' are not supported in General I/O WHERE clauses. EQ and NE are used instead.

Input Parameters

- G_IO_libraryName - Library or schema name.
- G_IO_tableName - Database table name.

Input and Output Parameters

G_IO_Result_Table Result - SAS Decision Services table. On input, this table contains column definitions (name and type). The specified columns are selected from the database, and coerced to the specified type if possible. On output, this table contains the original column definitions plus rows of data that are selected from the database.

Insert

Method name: SCInsertIntoTable.

Input Parameters

- G_IO_libraryName - Library or schema name.
- G_IO_tableName - Database table name.
- G_IO_Insert_Values - A SAS Decision Services table that contains multiple rows. Corresponding rows are inserted in the database table. Columns that occur in the database but not in this table are set to null or missing.

Input and Output Parameters

None.

Update

Method name: SCUUpdateTable.

Properties

G_IO_WHERE_Clause - WHERE clause. The WHERE clause property is a static string that is set on the General I/O Activity instance when it is inserted into a flow.

A WHERE clause is a SAS Decision Services (not SQL) Boolean expression.

Logical (AND, OR, NOT), relational (EQ, NE, GT, GE, LT, LE), and arithmetic (+, -, /, *) operators can be used. Here is an example: CustomerInfo.Income GT 50000.0.

As in a DATA step, a . (period) denotes a missing value.

Note: To ignore trailing blanks, use the operators

SQL_EQ,SQL_NE,SQL_GT,SQL_GE,SQL_LT,SQL_LE. These operators are valid only for strings.

Process parameters can be referenced as `:{Process parameter name}`. Here is an example: `CustomerInfo.LastName EQ :PV_CustomerLastName`

Note: '=' and '!=' are not supported in General I/O WHERE clauses. EQ and NE are used instead.

Input Parameters

- `G_IO_libraryName` - Library or schema name.

If this parameter is blank, the default database schema is used. The JDBC Connection resource that is specified in the General I/O activity definition is used.

Otherwise, if a JDBC library resource that has the given name is found, that resource is used to get the database schema name and JDBC Connection resource name. If the schema name in the resource is blank, the default database schema is used.

If a JDBC library resource with a given name is not found, the name is interpreted directly as a database schema name. The JDBC Connection resource that is specified in the General I/O activity definition is used.

Before SAS Decision Manager 5.5, this parameter specified a SAS libref. This name did not correspond to an actual database schema name. If your installation is earlier than 5.5, it can retain this name, but must add a JDBC library resource that has the same name. That resource can specify the database schema name.

- `G_IO_tableName` - Database table name.

A table name in the database schema (default or specific) that is specified by this `G_IO_libraryName`.

- `G_IO_Update_Values` - A SAS Decision Services table that contains one row. The table contains column definitions along with their corresponding values.

Output Parameters

`G_IO_Rows_Updated` - The number of database rows that are updated.

Insert Update

Method name: `InsertUpdateTable`

Properties

G_IO_WHERE_Clause - WHERE clause. The WHERE clause property is a static string that is set on the General I/O Activity instance when it is inserted into a flow.

A WHERE clause is a SAS Decision Services (not SQL) Boolean expression.

Logical (AND, OR, NOT), relational (EQ, NE, GT, GE, LT, LE), and arithmetic (+, -, /, *) operators can be used. Here is an example: CustomerInfo.Income GT 50000.0.

As in a DATA step, a . (period) denotes a missing value.

Note: To ignore trailing blanks, use the operators

SQL_EQ,SQL_NE,SQL_GT,SQL_GE,SQL_LT,SQL_LE. These operators are valid only for strings.

Process parameters can be referenced as *:{Process parameter name}*. Here is an example: CustomerInfo.LastName EQ :PV_CustomerLastName

Note: '=' and '!=' are not supported in General I/O WHERE clauses. EQ and NE are used instead.

Input Parameters

- **G_IO_libraryName** - Library or schema name.

If this parameter is blank, the default database schema is used. The JDBC Connection resource that is specified in the General I/O activity definition is used.

Otherwise, if a JDBC library resource that has the given name is found, that resource is used to get the database schema name and JDBC Connection resource name. If the schema name in the resource is blank, the default database schema is used.

If a JDBC library resource with a given name is not found, the name is interpreted directly as a database schema name. The JDBC Connection resource that is specified in the General I/O activity definition is used.

- **G_IO_tableName** - Database table name.

A table name in the database schema (default or specific) that is specified by this **G_IO_libraryName**.

- **G_IO_Update_Values** - A SAS Decision Services table that contains one row. The table contains column definitions along with their corresponding values.

- **G_IO_Increment_Values** - A SAS Decision Services table that contains one row. The table contains column definitions along with their corresponding values. The increment columns must be numeric.
- **G_IO_Insert_Values** - A SAS Decision Services table that contains one row. The table contains column definitions along with their corresponding values.

Output Parameter

G_IO_Rows_Updated - The number of database rows that are updated.

Increment Update

Method name: IncrementUpdateTable

Properties

G_IO_WHERE_Clause - WHERE clause. The WHERE clause property is a static string that is set on the General I/O Activity instance when it is inserted into a flow.

A WHERE clause is a SAS Decision Services (not SQL) Boolean expression.

Logical (AND, OR, NOT), relational (EQ, NE, GT, GE, LT, LE), and arithmetic (+, -, /, *) operators can be used. Here is an example: CustomerInfo.Income GT 50000.0.

As in a DATA step, a . (period) denotes missing.

Note: To ignore trailing blanks, use the operators

SQL_EQ,SQL_NE,SQL_GT,SQL_GE,SQL_LT,SQL_LE. These operators are valid only for strings.

Process parameters can be referenced as *:{Process parameter name}*. Here is an example: CustomerInfo.LastName EQ :PV_CustomerLastName

Note: '=' and '!=' are not supported in General I/O WHERE clauses. EQ and NE are used instead.

Input Parameters

- **G_IO_libraryName** - Library or schema name.

If this parameter is blank, the default database schema is used. The JDBC Connection resource that is specified in the General I/O activity definition is used.

Otherwise, if a JDBC library resource that has the given name is found, that resource is used to get the database schema name and JDBC Connection

resource name. If the schema name in the resource is blank, the default database schema is used.

If a JDBC library resource with a given name is not found, the name is interpreted directly as a database schema name. The JDBC Connection resource that is specified in the General I/O activity definition is used.

- **G_IO_tableName** - Database table name.

A table name in the database schema (default or specific) that is specified by this **G_IO_libraryName**.

- **G_IO_Update_Values** - A SAS Decision Services table that contains one row. The table contains column definitions along with their corresponding values.
- **G_IO_Increment_Values** - A SAS Decision Services table that contains one row. The table contains column definitions along with their corresponding values. The increment columns must be numeric.
- **G_IO_Rows_Updated** – The number of database rows that are updated.

Delete

Method name: DeleteFromTable

Properties

G_IO_WHERE_Clause - WHERE clause. The WHERE clause property is a static string that is set on the General I/O Activity instance when it is inserted into a flow.

A WHERE clause is a SAS Decision Services (not SQL) Boolean expression.

Logical (AND, OR, NOT), relational (EQ, NE, GT, GE, LT, LE), and arithmetic (+, -, /, *) operators can be used. Here is an example: CustomerInfo.Income GT 50000.0.

As in a DATA step, a . (period) denotes a missing value.

Note: To ignore trailing blanks, use the operators

SQL_EQ,SQL_NE,SQL_GT,SQL_GE,SQL_LT,SQL_LE. These operators are valid only for strings.

Process parameters can be referenced as *:{Process parameter name}*. Here is an example: CustomerInfo.LastName EQ :PV_CustomerLastName

Note: '=' and '!=' are not supported in General I/O WHERE clauses. EQ and NE are used instead.

Input Parameters

- **G_IO_libraryName** - Library or schema name.

If this parameter is blank, the default database schema is used. The JDBC Connection resource that is specified in the General I/O activity definition is used.

Otherwise, if a JDBC library resource that has the given name is found, that resource is used to get the database schema name and JDBC Connection resource name. If the schema name in the resource is blank, the default database schema is used.

If a JDBC library resource with a given name is not found, the name is interpreted directly as a database schema name. The JDBC Connection resource that is specified in the General I/O activity definition is used.

- **G_IO_tableName** - Database table name.

A table name in the database schema (default or specific) that is specified by this **G_IO_libraryName**.

- **G_IO_Rows_Deleted** - (Optional) The number of database rows that are deleted.

Library Resources

Using a library resource with the General I/O activity provides a level of indirection to the physical database schema name. It also provides a single location to specify the JDBC Connection resource name for a given schema.

The JDBC Connection resource provides database connection information. The resource name can be specified in a JDBC library resource. If a library resource is not used, the connection resource name is retrieved from the resource that is specified in the General I/O activity definition.

Guidelines for Creating Activities

Date and Time Formats That Are Supported by SAS Decision Services

SAS Decision Services 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 that represents 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 Decision Services supports a single datetime data type. When datetime values are passed from SAS Decision Services 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 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 is displayed incorrectly. Always use a DATETIME format to view such columns.

CAUTION! SAS Decision Services interprets DateTime values that are stored in a database or in SAS data sets as being in the Greenwich Mean Time (GMT) time zone. A default time zone (GMT) is associated with values that are read from a database because time zone information is not persisted with the data. As a best practice, always store DateTime values using the GMT time zone. This practice enables compatibility with SAS Decision Services and removes ambiguity.

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, locking errors can occur if you try to use General I/O to insert records into a SAS data set or to update records in a SAS data set. If concurrent writes are required, then use a database table.

If a data set is opened in an interactive SAS session while SAS Decision Services is reading the data set, locking errors occur. The errors occur because SAS locks the file when it is opened. It is recommended that all other SAS data sets be closed in an interactive SAS session while SAS Decision Services is using the SAS data set.

6

Integration

<i>Web Service Integration</i>	119
Overview	119
Web Service Definition Language	120
<i>WS-Security Integration</i>	123
Overview	123
Implementation	124
Configuration	125
Tools	131
<i>Integration with SAS Model Manager</i>	132
About SAS Model Manager	132
Best Practices	134
<i>REST API</i>	135

Web Service Integration

Overview

External applications see the SAS Decision Services engine server as a web service endpoint. They request decisions by sending web service requests to SAS Decision Services. When the endpoint is triggered by a Simple Object Access Protocol (SOAP) event request, the web service maps the incoming request to a SAS Decision Services

event object. It then passes it to the run-time engine for processing. After the run-time engine has completed its processing, a SOAP response is sent back to the invoking client.

One-way event operations are also supported that do not follow the common request and response message exchange pattern that is described above. In this case, a client sends a request and does not expect a response. Specifically, SAS Decision Services 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 that are defined in the request. The ordering of reply variables is also independent of the ordering of variables in the reply section of the event definition. Client applications should not rely on reply variables being returned in any particular order.

Web Service Definition Language

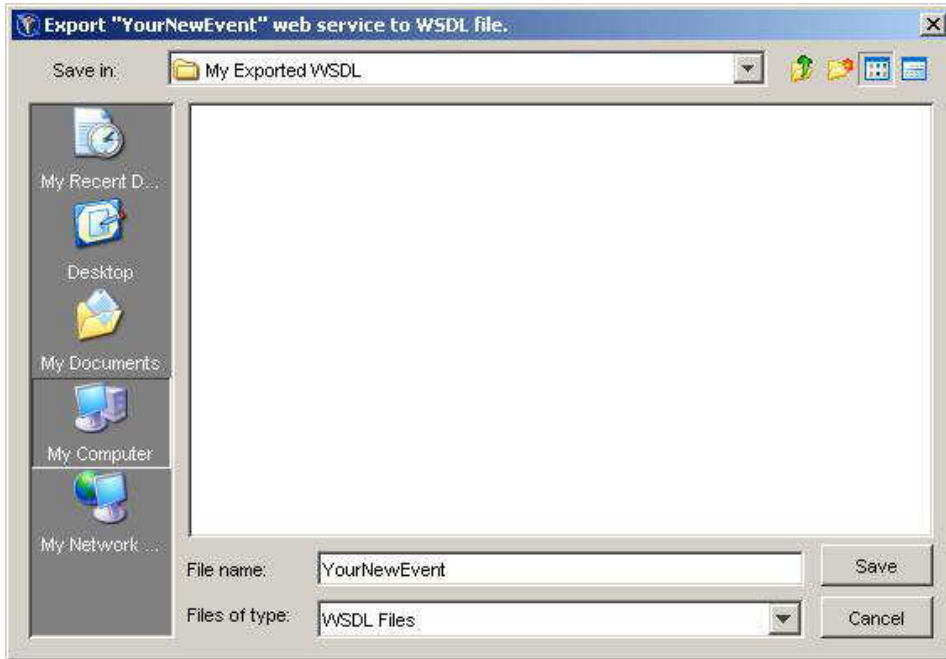
Retrieve a WSDL File

You can retrieve the Web Service Definition Language (WSDL) file for a given SAS Decision Services event as follows:

- 1 Open SAS Management Console.
- 2 On the **Folders** tab, select **System ► Applications ► SAS Decision Services ► Decision Services 6.4**.
- 3 Navigate to the SAS Decision Services repository for which you want to generate a WSDL file.
- 4 Right-click the event in the repository and click **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 Navigate to a location to store and name the WSDL file.

Figure 6.1 WSDL File

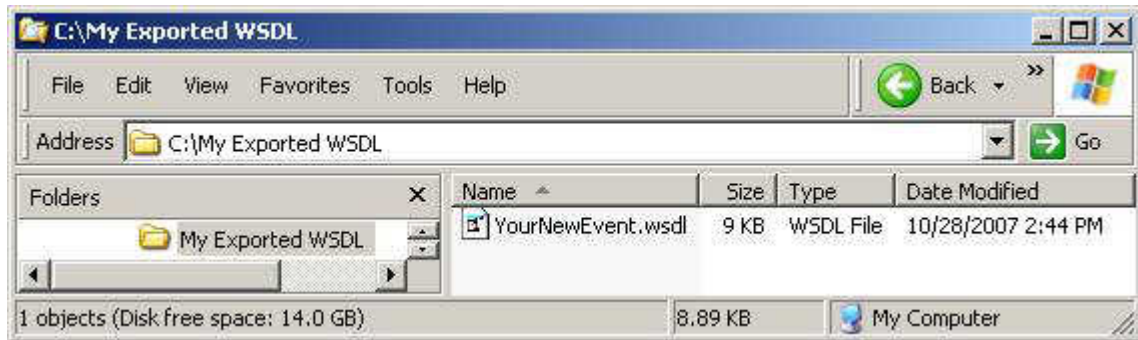


- 7 Click **Save**.

Figure 6.2 WSDL Created



Verify that the new WSDL exists by browsing the directory to locate the file.

Figure 6.3 Exported WSDL

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 Decision Services engine server. After the decision flow is activated, the event 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</rdm: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>

```

Because requests to SAS Decision Services can originate from multiple time zones, ClientTimeZoneID is a required field in the SAS Decision Services header. Time zone names from the public domain time zone (TZ) database are accepted. The public domain time zone database is maintained by the Internet Assigned Numbers Authority, available at <http://www.iana.org/time-zones>.

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

WS-Security Integration

Overview

WS-Security secures the message transmission between the client application and the SAS Decision Services engine. The use of WS-Security with SAS Decision Services is optional. Any of the three implemented aspects of WS-Security can be used alone or can be specified in conjunction with any combination of the other aspects. The following aspects have been implemented:

Timestamp

The message is marked by the sender with creation and expiry timestamps, which the receiver validates. The SAS Decision Services web service can be configured to

validate the request message with the expiry timestamp, as well as an offset from its own clock. It can also set timestamps on the reply message. This mechanism is used to prevent replay or "man-in-the-middle" attacks.

Signature

Message signing is implemented by the sender signing the message using its private key and the receiver decrypting it using the trusted or public key of the sender's key. This is true for both request and response messages. The server needs to access the trusted key of the client's private key, and the client needs access to the trusted key of the server's private key. Frequently, the public keys might have to be certified by a certificate authority.

Encryption

This is implemented by using a symmetric key that travels with the message. The key is encrypted by the sender, using the trusted key of the receiver (opposite of signing). Like with the signature, this mechanism is true for both request and response messages. The sender can send only to a receiver whose trusted key is available to the sender. All passwords can also be sas002 encoded.

Implementation

The SAS Decision Services web service is implemented as a Java web application. WS-Security is implemented using Apache WSS4J. The WS-Security implementation can be configured and customized by setting the appropriate values for system properties. Not all features of Apache WSS4J are exposed or configurable.

To configure some of the features, private and trusted keys are required. These keys are held in key stores. Sometimes it is convenient to hold the private and trusted keys in separate stores. A key store holding only trusted keys is also known as a trust store. The JRE implementation contains a trust store called CACerts that is used by default.

As part of setting up WS-Security, public and private keys must be created, certified by certificate authorities, and distributed among the client and server key stores, as per your IT policies.

Configuration

The following system properties can be used to configure the WS-Security implementation in the SAS Decision Services engine web service:

Note: All passwords can be sas002 encoded.

Category	Description	Property	Default Value
Signature Key Store	The key store containing the key used for signing outgoing messages.		Points to the CACert in JRE.
	Password to access this key store.	sasds.ws-security.signatureKeyStore.password	changeit
	Location of this key store.	sasds.ws-security.signatureKeyStore.location	file:\${java.home}/lib/security/cacerts
Signature Trust Store	The key store containing trusted certificates for verifying signed incoming messages.		Points to the CACert in the JRE .
	Password to access this key store.	sasds.ws-security.signatureTrustStore.password	changeit
	Location of this key store.	sasds.ws-security.signatureTrustStore.location	file:\${java.home}/lib/security/cacerts
Encrypt Key Store	The key store containing key used for decrypting incoming messages.		Points to the CACert in the JRE.
	Password to access this key store.	sasds.ws-security.encryptKeyStore.password	changeit

	Location of this key store.	sasds.ws-security.encryptKeyStore.location	file:\${java.home}/lib/security/cacerts
Encrypt Trust Store	The key store containing the key used for encrypting outgoing messages.		Points to the CACert in the JRE.
	Password to access this key store.	sasds.ws-security.encryptTrustStore.password	changeit
	Location of this key store.	sasds.ws-security.encryptTrustStore.location	file:\${java.home}/lib/security/cacerts
General	The general WS-Security properties.		
	Sets the name of the validation actor. Actors are subsystems that process the SOAP message with a specific purpose.	sasds.ws-security.validationActor or	
	The actor name of the wsse:Security header. If this parameter is omitted, the actor name is not set. The value of the actor or role has to match the receiver's setting or can contain standard values.	sasds.ws-security.securementActor	
	Enables the mustUnderstand attribute on WS-Security headers on outgoing messages.	sasds.ws-security.securementMustUnderstand	True
Timestamp	The properties used for time-stamping the messages or for validating the timestamp on a message.		

	Determines whether outbound timestamps have precision in milliseconds.	sasds.ws-security.timestampPrecisionInMilliseconds	True
	Determines whether to enable strict timestamp handling. If this is true, then use the validationTimeToLive to determine whether a message is expired. Otherwise, use the expiry timestamp on the message.	sasds.ws-security.timestampStrict	False
	The time difference between creation and expiry time in seconds in the WS-Security timestamp of the outbound message.	sasds.ws-security.securementTimeToLiveInSeconds	300
	Determines whether to enable strict timestamp handling. If this is true, then use the validationTimeToLive to determine whether a message is expired. Otherwise, use the expiry timestamp on the message.	sasds.ws-security.validationTimeToLiveInSeconds	300
Signature	The properties that control validating the signature on an inbound message as well as creating a signature on an outbound message.		
	Whether to enable signature confirmation.	sasds.ws-security.enableSignatureConfirmation	False
	The alias name of the private key used to sign the outbound message.	sasds.ws-security.securementUsername	

The alias name of the private key used to sign the outbound message. If both this value and sasds.ws-security.securementUsername are set, this value prevails.	sasds.ws-security.securementSignatureUser	
The password of the private key used to sign the outbound message.	sasds.ws-security.securementPassword	
Describes how the key is referenced in a signed or encrypted message header. Valid values are: <ul style="list-style-type: none"> ■ IssuerSerial ■ DirectReference ■ X509KeyIdentifier ■ Thumbprint ■ SKIKeyIdentifier ■ KeyValue (signature only) ■ EncryptedKeySHA1 (encryption only) For certificate authentication use DirectReference.	sasds.ws-security.securementSignatureKeyIdentifier	IssuerSerial
Defines what signature algorithm to use. The default is set by the data in the certificate, such as one of the following: <ul style="list-style-type: none"> ■ http://www.w3.org/2000/09/xmlsig#rsa-sha1 ■ http://www.w3.org/2000/09/xmlsig#dsa-sha1 	sasds.ws-security.securementSignatureAlgorithm	The default is set by the data in the certificate.
Parameter to define what parts of the request should be signed.	sasds.ws-security.securementSignatureParts	The SOAP body is signed by default.

Encrypt	The properties that control the decryption of inbound and outbound messages.	
	<p>Two values are possible:</p> <p><code>embeddedEncryptedSymmetricKeyValidationHandler</code> The symmetric key for encryption is included in the message received.</p> <p><code>embeddedKeyNameValidationHandler</code> Only the name of the key is available in the message.</p> <p>In both cases, use the next property to retrieve the private key for decrypting.</p>	<p><code>sasds.ws-security.validationEncryptionHandler</code></p> <p><code>embeddedEncryptedSymmetricKeyValidationHandler</code></p>
	The password for the private key used to decrypt the inbound message.	<code>sasds.ws-security.validationEncryptPrivateOrSymmetricKeyPassword</code>
	The user name for encryption. The encryption function uses the public key of this user's certificate to encrypt the generated symmetric key. If only the encryption of the SOAP body data is requested, it is recommended to use this parameter to define the user name.	<p><code>sasds.ws-security.securementEncryptionUser</code></p> <p>If this parameter is not set, then the encryption function falls back to the <code>sasds.ws-security.securementUsername</code> property to get the certificate.</p>
	The text of the key name to be sent for encryption in the <code>KeyInfo</code> .	<code>sasds.ws-security.securementEncryptionEmbeddedKeyName</code>

<p>Defines what key identifier type to use. The WS-Security specifications recommends that you use the identifier type IssuerSerial. Possible values are:</p> <ul style="list-style-type: none"> ■ IssuerSerial ■ X509KeyIdentifier ■ DirectReference ■ Thumbprint ■ SKIKeyIdentifier ■ EmbeddedKeyName 	<p>sasds.ws-security.securementEncryptionKeyIdentifier</p>	<p>IssuerSerial</p>
<p>Defines what algorithm to use to encrypt the generated symmetric key. Currently Apache WSS4J supports:</p> <ul style="list-style-type: none"> ■ http://www.w3.org/2001/04/xmlenc#rsa-1_5 ■ http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p 	<p>sasds.ws-security.securementEncryptionKeyTransportAlgorithm</p>	<p>http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p</p>
<p>Defines what symmetric encryption algorithm to use. Apache WSS4J supports the following algorithms:</p> <ul style="list-style-type: none"> ■ http://www.w3.org/2001/04/xmlenc#tripleDES-cbc ■ http://www.w3.org/2001/04/xmlenc#aes128-cbc ■ http://www.w3.org/2001/04/xmlenc#aes256-cbc ■ http://www.w3.org/2001/04/xmlenc#aes192-cbc <p>Except for http://www.w3.org/2001/04/xmlenc#aes192-cbc, all of these algorithms are required by the XML encryption specification.</p>	<p>sasds.ws-security.securementEncryptionSymAlgorithm</p>	<p>http://www.w3.org/2001/04/xmlenc#aes128-cbc</p>

	Property to define what parts of the request should be encrypted. For more information, refer to Apache WSS4J documentation.	sasds.ws-security.securementEncryptionParts	If no list is specified, the handler encrypts the SOAP body in Content mode by default.
Actions	These are space-separated lists of tokens that define the steps for incoming (validation) or outgoing (securement) messages. The valid tokens SAS Decision Services supports are NoSecurity, Timestamp, Signature, and Encrypt. The order in which these are applied is important and must match the sender's or receiver's order for validation and securement respectively.		
		sasds.ws-security.validationActions	NoSecurity
		sasds.ws-security.securementActions	NoSecurity

These properties can be set by defining them in the SAS Management Console Configuration Manager plug-in for SAS Decision Services. In SAS Management Console, locate the properties for SAS Decision Services Engine under **Application Management ► Decision Services Engine Server 6.4**. On the **Advanced** tab, add or update the properties that you need to set. For example, you could define a property with a name of `sasds.ws-security.securementEncryptionKeyIdentifier` and a value of `X509KeyIdentifier`

Tools

For most implementations, it is required to create key stores with private and public key pairs in them, and then distribute them in other key stores. KeyTool is a command-line

utility distributed with a JRE. Here are some common commands that are useful when setting up key stores for WS-Security:

```
keytool -genkeypair -alias <key name> -keyalg <key algorithm> -sigalg <signature algorithm> -validity <days> -keystore <key store name>
```

Generates a private and public key pair in the key store. It also creates a new key store if it does not exist.

```
keytool -list -v -keystore <key store name>
```

Lists keys in the key store.

```
keytool -export -alias <key name> -keystore <key store name> -rfc -file <certificate file name>
```

Exports the public key from the key store as a self signed certificate, for import into a trust store.

```
keytool -certreq -alias <key name> -keystore <key store name> -file <certificate request file name>
```

Generates a certificate request for sending to a certificate authority.

```
keytool -import -alias <key name> -file <certificate file name> -keystore <trust store name>
```

Imports the certificate into the trust store.

Integration with SAS Model Manager

About SAS Model Manager

Directory Lockdown

You can limit the reach and activities of a SAS server by putting it in a locked-down state. When running SAS in a locked-down state, access to files and directories on the host system is restricted to a lockdown path list (a specified list of paths and files that are valid for the SAS server to access).

SAS Decision Services requires that the `<SAS_Configuration_Directory>/Applications/SASDecisionServicesServerConfig/DS2Conversion/design`

directory be added to the lockdown path list by the SAS administrator. The location of the directory is configurable. However, the specific configured location can be found on the **Advanced** tab of the Decision Services Design Middle Tier Properties dialog box, as the value of the `sasds.designserver.ds2.generation.folder.path` property.

Note: The scoring activity publishing will fail if the directory is not added to the lockdown path list. The directory is not added to the list by default.

You can lock down at the SAS Application Server level as well as the Pooled Workspace Server level. By default, the Pooled Workspace Server is configured to run under SASApp, and the value is SASApp - Logical Pooled Workspace Server. The specific name of your Pooled Workspace Server can be found on the **Advanced** tab of the Decision Services Design Middle Tier Properties dialog box, as the value of the `sasds.designserver.logical.workspace.server.name` property.

It is recommended that you lock down at the higher SAS Application Server level, which locks down all servers under SASApp (such as the Logical Pooled Workspace Server and the Logical Stored Process Server). Then, apply the lockdown path list at the specific Pooled Workspace Server level. This is the most restrictive option and does not allow other servers under SASApp to write to the accessible folders that are meant for the Pooled Workspace Server.

The lockdown can be effected by modifying the `sasv9_usermods.cfg` file (found at `<sas-configuration-directory>/Lev1/SASApp/`) by specifying the following option:

```
-lockdown
```

To specify what folders the SAS server can write to, modify the `autoexec_usermods.sas` file by creating (or adding the folder to) an entry in the following form:

```
lockdown path=
"<sas-configuration-directory>\Lev1\Applications\SASDecisionServicesServerConfig\
DS2Conversion\design";
```

For specific instructions on lockdown, and to create or append a list of folders that the SAS server can write to, see the *SAS Intelligence Platform: Security Administration Guide*.

Overview

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

See the SAS Model Manager documentation for information. This section describes the integration and interoperability between SAS Decision Services and SAS Model Manager.

Scoring models are converted into SAS activities using the DSTRANS procedure. PROC DSTRANS was created to convert into DS2 code those models that SAS Enterprise Miner produced. DSTRANS is limited to a subset of SAS DATA step functionality. See PROC DSTRANS in the *Base SAS Procedures Guide*.

The development environment enables a user to choose any of the scoring projects that have been published to SAS Real-Time Decision Manager by SAS Model Manager. After conversion to a SAS activity through the Customer Intelligence plug-in for SAS Management Console, a scoring project can be added to a decision flow in multiple places, allowing multiple models to be included in a single decision flow.

Best Practices

- One SAS Metadata Repository folder for publishing models should be created for each development, test, and production SAS Decision Services environment in your deployment.
- A scoring project should be published to the development folder first and tested in the SAS Decision Services development environment.
- Using this practice, the same testing, approval, and promotion policies that are applied to decision flows can be applied to scoring projects.

REST API

For REST API information, see [Appendix 5, “REST API for SAS Decision Services Transaction Processing,”](#) on page 245 or [Appendix 6, “SAS Decision Services Administration API,”](#) on page 275.

7

Installation

Overview	138
Choosing Environments	138
Dependent SAS Products	139
SAS Web Application Server	139
SAS Web Infrastructure Platform Data Server	139
SAS BI Web Services for SAS 9.4	139
SAS Authentication Server	139
SAS Federation Server	140
DataFlux Secure	140
Best Practices for SAS Decision Services	
Deployment Scenarios	140
Overview	140
Best Practices for SAS Decision Services	
Performance and High Availability	142
Deployment Scenarios	143
SAS Federation Server Connection and	
Statement Pool Tuning	148
Configuring SAS Decision Services	149
Deploying and Starting SAS Decision Services	
Web Applications in a Cluster	149
Understanding Clusters	149
Adding a Vertical Cluster Member	150

Adding a Horizontal Cluster Member	151
Configuring Monitoring for Additional Engine Nodes	153
<i>Rebuilding SAS Decision Services Design, Engine, and Monitor Server Web Applications</i>	154
<i>Rebuilding the SAS Web Application Server Configurations</i> ..	155
<i>Post-Installation Reconfiguration</i>	155
Engine and Design Server Reconfiguration	155

Overview

Before installing SAS Decision Services, work with your on-site SAS support personnel to determine the hardware, network, and software topology that your throughput and response time require.

Choosing Environments

At a minimum, install one development and one production environment. You can install one or more test environments, depending on your organization's testing policies. Decision flows can be unit tested in the development environment. A test environment is used to test decision flows in an environment that is similar to production. The test and production environments have only a few differences:

- The test environment is not connected to live channels or customer-facing systems.
- More hardware and network resources might be allocated to the production environment.

The development environment is typically not clustered. The production environment might use a clustered middle tier, database tier, and SAS Federation Server tier.

Dependent SAS Products

SAS Web Application Server

SAS Web Application Server is a lightweight server that provides enterprise-class features for running SAS web applications. The server is based on VMware vFabric tc Server. By packaging the server and software that can automate server configuration tasks, SAS simplifies the demands for managing a web application server. For more information, see *SAS Intelligence Platform Middle-Tier Administrator's Guide*

SAS Web Infrastructure Platform Data Server

SAS Web Infrastructure Platform Data Server is used to store the monitoring data that is collected during real-time and batch execution of flows on the engine server.

SAS BI Web Services for SAS 9.4

SAS BI web services for SAS 9.4 enables you to select a set of stored processes in SAS Management Console and use the Web Service Maker to deploy them as web services. The Web Service Maker generates a new web service that contains one operation for each stored process that you selected. For more information about developing web services, see *SAS BI Web Services Developer's Guide*.

To invoke a SAS BI web service from SAS Decision Services, include a web service activity in your decision flow. SAS BI web services are useful if you want to execute DATA or PROC steps, or if you want to use SAS macro code. However, keep in mind that these code constructs carry significant performance penalties.

SAS Authentication Server

SAS Authentication Server provides security for the DS2 programs and data that is accessed by SAS Federation Server.

SAS Authentication Server is required in all deployments that include a SAS Federation Server.

For more information, see *SAS Authentication Server Administrator's Guide* that is located in the `doc` folder of your authentication server installation directory.

SAS Federation Server

The SAS Federation Server is a compute server that executes SAS Decision Services activities that are written in the DS2 programming language.

For more information, see *SAS Federation Server Administrator's Guide* that is located in the `doc` folder of your SAS Federation Server installation directory.

SAS Federation Server Manager is used to configure and manage the SAS Federation Server DSNs and data services. SAS Data Management Studio for Federation Server is used to add users and groups to the authentication server.

DataFlux Secure

If you use SAS/SECURE for your SAS servers, then you must license DataFlux Secure. The reason is that encryption must be set consistently across all SAS server components. For example, if you use AES encryption for the metadata server, then all SAS servers must be configured with AES encryption.

Best Practices for SAS Decision Services Deployment Scenarios

Overview

Decision services consist of processing steps (called activities) and conditional control logic. The conditional logic determines which activities are executed and in what order. The path of execution through a decision service is typically influenced by the input data and by the results of each processing step. The response time for a single execution of

a decision service is the sum of the latencies of the processing steps along the path of execution.

Because paths of execution are data dependent, a single decision service might exhibit a range of latencies. Furthermore, multiple heterogeneous decision services can be deployed at the same time, each consuming a portion of the available computational resources on a given server. It is often difficult to anticipate all possible combinations of data and their influences on performance, making hardware capacity planning a challenge.

Therefore, it is a good practice to create a baseline system to deploy your decision services into. You would also use it to measure performance against historical data, and to extrapolate the results to create a hardware plan that meets your throughput and latency requirements.

The following sections explain how to create an appropriate baseline SAS Decision Services environment. The SAS Decision Services environment can be used to collect the data necessary to plan the production hardware capacity that is required for your own unique set of decision services.

Note: If you plan to use SAS Real-Time Decision Manager treatment sets, then hardware capacity planning must be a critical component of your planning process. This is due to the computationally intensive nature of the treatment set application. Actual performance depends on several factors. For more information, see the SAS Real-Time Decision Manager documentation.

Component	Used By	Performance Critical?	Tier
SAS Management Console	Design and Run time	No	Client
SAS Decision Services HQ Plug-in for SAS Environment Manager	Run time	No	Middle
SAS Metadata Server	Design and Run time	No	SAS

Third-party Database Management System	Design and Run time	Yes	Database (DBMS)
SAS Decision Services 6.4 Design Server	Design	No	Middle
SAS Decision Services 6.4 Engine	Run time	Yes	Middle
SAS Decision Services 6.4 Monitor	Run time	No	Middle
SAS Federation Server 4.1	Design and Run time	Yes	Compute
SAS Authentication Server 4.1	Design and Run time	No	Compute
SAS Data Management Studio 2.5M1	Design and Run time	No	Client
SAS Object Spawner	Run time	No	SAS

Best Practices for SAS Decision Services Performance and High Availability

- SAS Decision Services has a design environment and a run-time environment. The design environment is used for developing, modifying, and functional testing of decision services. The run-time environment is used for production. A run-time environment can also be used for integration or performance testing. However, if a testing environment is deployed, it should be separate from the production environment. The production environment should always be dedicated to production processing only. High performance, measured in terms of throughput and latency, and around-the-clock availability are typically critical for run-time environments and less important for design environments.
- The following components are critical to run-time performance and availability:

- engine
 - SAS Federation Server
 - database management system
- In the run-time environment, the engine, SAS Federation Server, and database instances should be installed on dedicated hardware. Service levels cannot be guaranteed if external software is allowed to consume resources. Batch processes and online transaction processing should never share hardware resources.
 - The SAS Federation Server has approximately twice the throughput capability as the engine. Therefore, an optimized deployment should assign more powerful hardware to the engines than to SAS Federation Server.
 - The numbers of servers that are allocated to the middle, SAS, and database tiers should be proportional to your throughput and latency requirements. At least two servers in each tier are required to support failover and ensure high availability.
 - The engine and SAS Federation Server tiers can either be colocated or dislocated, as necessary. See your on-site SAS support personnel for advice regarding which configuration is more appropriate for your specific deployment.
 - SAS Metadata Server, SAS Management Console, and the object spawner have a minimal impact on performance because they are not directly involved in transaction processing.

Deployment Scenarios

Easy Button

An "easy button" deployment is a deployment where the default settings, where available, are used during the installation and configuration process. This results in a design-time system and a run-time system. Easy button deployments are suitable for decision service design and functional testing, but are appropriate only for production use in cases where high performance and high availability are not required.

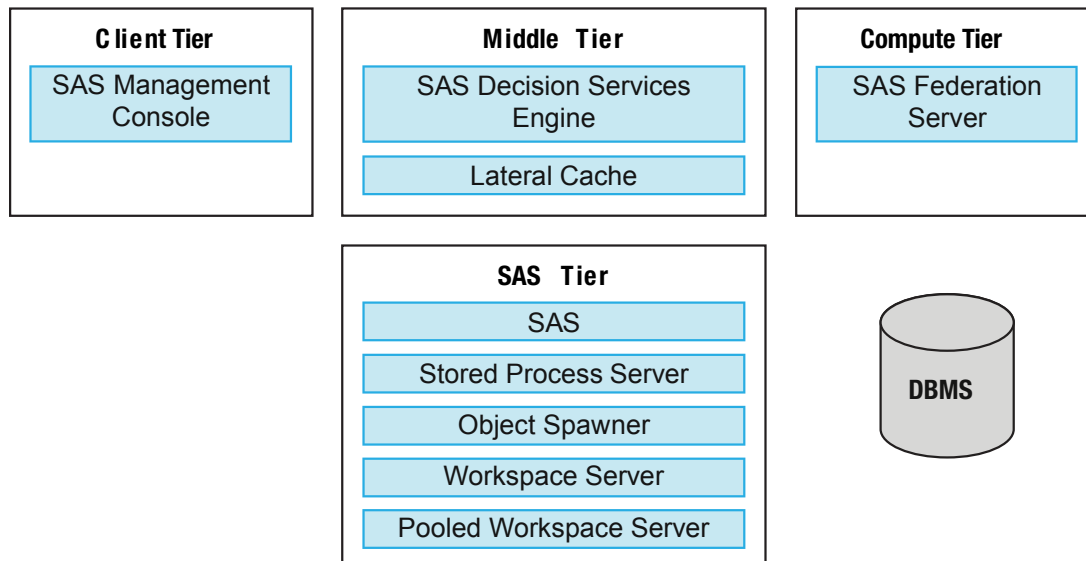
The design-time system contains a design server for creating and modifying decision services. It contains many of the same software components as a production system, in order to enable functional testing of decision services. A major difference between a

design environment and a production environment is that a production deployment typically includes load-balanced, clustered engine servers and multiple SAS Federation Server instances for scalability and high availability.

Production Deployments

SAS Decision Services production deployments consist of the following major components:

- SAS Decision Services engine server cluster (at least two engine servers are required for high availability)
- one or more SAS Federation (TKTS) Server and SAS Authentication Server pairs (at least two SAS Federation Servers are required for high availability)
- SAS Management Console plug-in, for centralized control and monitoring
- SAS Web Server or a third-party load balancer
- third-party database management system, clustered for high availability
- workspace server, for publishing activities and other DS2 assets to SAS Federation Server
- (Optional) SAS Stored Process Server for the execution of BI web services

Figure 7.1 Production Deployment

Here are examples of the factors that actual hardware capacity planning depends on:

- peak transaction volume
- maximum latency requirements
- minimum throughput requirements
- business logic complexity
- analytic complexity
- size of request and response messages
- amount and frequency of disk I/O
- if you are deploying Real-Time Decision Manager treatment campaigns, the number of treatments, treatment sets, custom user-defined fields, and custom user-defined field lists
- external system dependencies, such as external web service calls made by a decision service

Scenario: Complex Business Logic and Light Analytics

A typical SAS Decision Services scenario might include business logic combined with one or two high-performance predictive models that generate scores, such as propensity or risk. For the purposes of this scenario, assume that all required data is passed in to the decision service through the event. Therefore, no database I/O occurs. In such a scenario, processing is approximately evenly divided between the business logic and the analytics. In general, the business logic executes in the engine middle tier and the analytics execute inside SAS Federation Server.

Because a SAS Federation Server executes with approximately twice the throughput of the engine middle tier, a baseline topology might include a 16-core middle tier server and an 8-core SAS Federation Server. Alternatively, two engine servers could be allocated per SAS Federation Server, if all servers are equally powerful.

The baseline topology hardware should be multiplied until latency and throughput requirements are achieved.

A hardware failover capability requires at least two servers per tier. All SAS Federation Servers should have access to a common clustered database management system. This database should be clustered to support failover. A common database should be used to allow all DS2 activity packages to be accessed by all SAS Federation Servers. Using servers of equal capacity for this scenario, a system capable of hardware failover would have four middle tier servers, two SAS Federation Servers, and a database server cluster.

Although data I/O was not included in this scenario, a database management system might be used to store the activities, which are persisted in the database as DS2 packages. Alternatively, the DS2 packages can be stored in SAS data sets on a shared file system.

Scenario: Complex Business Logic and Complex Analytics

Another typical SAS Decision Services scenario includes both complex business logic and complex analytics, where three or more predictive models, or one or more very complex models, are used. In this scenario, the analytics require more processing cycles than the business logic.

Because a SAS Federation Server executes at approximately twice the speed of the engine middle tier and is doing twice as much work, a baseline topology might include a 16-core middle tier server and a 16-core SAS Federation Server.

The baseline topology should be multiplied until latency and throughput requirements are achieved.

A hardware failover capability requires at least two servers per tier. As mentioned earlier, all SAS Federation Servers should have access to a common clustered database management system. For this scenario, a typical system capable of hardware failover would have two middle tier servers, two SAS Federation Servers, a database server cluster, and SAS Web Server or a third-party front-end load balancer.

Real-Time Decision Manager Treatment Set Campaign with 100 Treatment Campaigns

Depending on a number of factors, such a deployment might require more than 100 cores in the production environment. This scenario should be planned with the assistance of SAS Professional Services. Treatment campaigns require special planning because they typically consume an order of magnitude more compute resources than advanced analytics, such as credit risk applications.

Heterogeneous Decision Service Considerations

In reality, most deployments include a mix of variations on the scenarios described earlier. To determine the proportion of processing cycles that are consumed by business logic versus analytics, consider the cumulative effects of each decision service, as well as the relative frequencies of the events that are bound to each. When measuring your baseline system using historical data, record CPU use for each server at several points during the simulation run. The results will indicate whether the processing load is balanced, and where adjustments to hardware resources must be made.

In addition to these examples, your client applications, such as SAS Real-time Decision Manager or SAS Decision Manager, can generate complex decision flows that address specific business problems. To accurately plan hardware capacity, you must understand the processing that is performed by the flows that you will be running, the expected transaction volumes, data requirements, and performance constraints.

Database I/O Considerations

Disk I/O is typically the most expensive operation that real-time systems perform. Therefore, database hardware capacity planning and performance tuning are critical to the performance of decision services that read from or write to disk. Contact your database management system vendor for guidance.

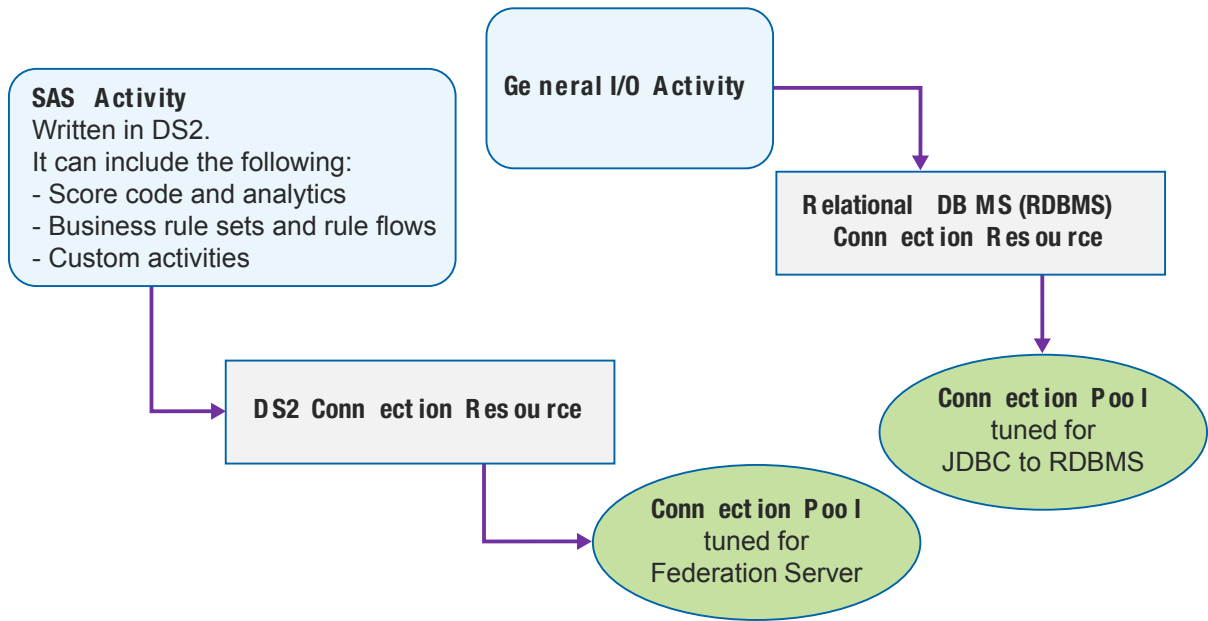
SAS Federation Server Connection and Statement Pool Tuning

When you are using 8-way to 16-way blade servers, the optimum connection pool and prepared statement pool sizes are usually both between 16 and 24 inclusive. A SAS Federation Server allocates a thread per JDBC Connection, so pool size has a direct effect on performance.

In general, processing that is highly CPU-bound performs best when the number of threads of execution in the SAS Federation Server is close to the number of cores per SAS Federation Server. Processing that is highly I/O bound benefits from more threads, so that there is always a thread ready to run whenever a running thread blocks to wait for I/O.

Your individual hardware might perform differently, so you might want to experiment with different pool sizes to achieve optimum performance. Pool tuning controls are contained by the JDBC System Resource, which can be accessed through SAS Management Console.

Note: The guidelines above apply only to the SAS Federation Server, which has relatively heavyweight threads. The engine middle tier uses Java threads, which are by comparison very lightweight. An engine server might run hundreds of these threads at a time.

Figure 7.2 Activities, Resources, and Connection Pools

Configuring SAS Decision Services

For more information about configuring SAS Decision Services, see *SAS Decision Services 6.4 Deployment Guide*.

Deploying and Starting SAS Decision Services Web Applications in a Cluster

Understanding Clusters

In order to provide greater scalability, availability, and robustness, SAS Application Server supports both vertical and horizontal clustering. With clustering, multiple server instances participate in a load-balancing scheme to handle client requests. Workload

distribution is managed by the SAS Web Server. SAS Web Server is configured as a load-balancing HTTP proxy.

The server instances in a cluster can coexist on the same machine (vertical clustering), or the server instances can run on a group of middle-tier server machines (horizontal clustering). The web applications can be deployed on both vertical and horizontal clusters.

The SAS Deployment Wizard deploys SAS Decision Services web applications to the web application server. However, you can also deploy web applications manually from the web application server. The web applications are in the `SAS-config-dir\Lev1\Web\Staging` directory.

There is no required start-up order for deploying the web applications.

- 1 SAS Decision Services Design Middle Tier
(sas.decisionservices.designserver6.4.ear)
- 2 SAS Decision Services Engine Server (sas.decisionservices.engine6.4.ear)

Adding a Vertical Cluster Member

Note: SAS Decision Services supports vertical member clustering if the servers are virtual or are locally partitioned.

Vertical clustering is the practice of deploying multiple identically configured web application server instances on a single machine. It can offer some improvement for availability. In the event that one web application server instance crashes (or an application on one server instance stops), the applications remain available on the other web application server instances.

To add a vertical cluster member:

- 1 Stop the web application server instance and other middle-tier servers.

```
SAS-config-dir\Lev1\Web\Scripts\AppServer\appsrvconfig.cmd stop
```


- 2** Locate the SAS software depot on the machine and start the SAS Deployment Wizard. When you start the SAS Deployment Wizard, specify your plan file or select the plan that you used from the list of standard plans.
- 3** When offered the choice to install and configure software, select the check box for configuring software, clear the check box for installing software, and click **Next**.
- 4** When you specify the configuration directory, the wizard provides a warning that the directory contains existing files. Click **Yes** to confirm the warning.
- 5** On the Select Products to Configure page, select the check box for **SAS Web Application Server Configuration** only and click **Next**.
- 6** On the Web Application Server: Managed Server Ports page, use the **Cluster Member Multiplier** menu to specify the number of web application server instances to configure.

For the pages before this one, and after it, specify the same values that were entered during the initial configuration.

- 7** Stop the middle-tier servers again (they were started when the SAS Deployment Wizard completed).

```
SAS-config-dir\Lev1\Web\Scripts\AppServer\appsrvconfig.cmd stop
```

- 8** Configure the SAS web applications and resources, such JDBC data sources and JMS queues.

```
SAS-config-dir\Lev1\Web\Scripts\AppServer\appsrvconfig.cmd -a
```

The configuration scripting tool (appsrvconfig.cmd) starts the servers when it completes.

TIP Log on to SAS Environment Manager and add the new servers to your inventory.

Adding a Horizontal Cluster Member

Horizontal clustering is the practice of deploying SAS Web Application Server instances on multiple machines. This can assist with improving performance and provide greater

availability to guard against hardware failure. In the event that one machine or web application server instance crashes (or an application on one server instance stops), the applications remain available on the other machines.

The SAS Deployment Wizard is used to add an additional middle-tier node. When it runs, it performs the following tasks:

- installs and configures a SAS Web Application Server instance
- configures SAS Web Server to load-balance HTTP requests to the new server instance
- starts the server instance

To add a horizontal cluster member:

- 1 On the machine that hosts the SAS Web Server, make sure the SAS Deployment Agent is running. The agent can be started from `SASHome\SASDeploymentAgent\9.4\agent.bat start`.

If the first instance of SAS Web Application Server is not installed on the same machine as SAS Web Server, then start the deployment agent on that machine too.

- 2 Copy the SAS software depot to the machine to use, or make sure the depot is available from a network share.
- 3 Start the SAS Deployment Wizard on the new machine to use. On the deployment step page, select **Middle Tier Node**.

Note: You can use the **Cluster Member Multiplier** menu on the Web Application Server: Managed Server Ports page to combine vertical clustering with horizontal clustering.

- 4 On the first web application server instance that was configured with the SAS Deployment Wizard, set the following JVM option when the SAS Deployment Wizard completes.

```
-Dcom.sas.server.isclustered=true
```

After you make this change, restart the web application server instance.

TIP Log on to SAS Environment Manager and add the new machine and servers to your inventory.

Configuring Monitoring for Additional Engine Nodes

You must configure monitoring for each additional engine node in the cluster. Information about engine nodes is stored in the `monitor.DCSV_TOPOLOGY` table of the Decision Services database.

In the `monitor.DCSV_TOPOLOGY` table, add the following IP address and port information for each additional engine cluster node that is deployed under `SASServer7`.

`engine_cluster_node_ip_address`

The actual IP address of the SAS Web Application Server that is deployed in the cluster.

`engine_cluster_node_port`

The actual port of the SAS Web Application Server that is deployed in the cluster.

Note: Do not specify the IP address of the SAS Web Server.

The following is an example of the SQL to use, which can be submitted through PSQL:

```
INSERT INTO monitor.DCSV_TOPOLOGY
values ('<engine_cluster_node_ip_address>', <engine_cluster_node_port>);
```

The PSQL commands can be executed on the SAS Web Infrastructure Platform Data Server: `<SAS_HOME>/SASWebInfrastructurePlatformDataServer/9.4/bin/psql`

Here is an example using PSQL:

1 Connect to the DecisionServices database:

```
psql -h <host> -p <port> -U DecisionServices DecisionServices
```

2 Enter the password for the DecisionServices database.

3 Insert information for each additional engine cluster node:

```
INSERT INTO monitor.dcsv_topology VALUES
('<engine_cluster_node_host>', <engine_cluster_node_port>);
```

Rebuilding SAS Decision Services Design, Engine, and Monitor Server Web Applications

The files for the SAS web applications are stored in the `SAS-config-dir\Lev1\Web\Staging` directory.

When the SAS Deployment Manager is used to rebuild a SAS Decision Services web application, the files for the web application in the previous directories are overwritten. The following table identifies the product configuration name that is used in the SAS Deployment Manager for the SAS Decision Services web applications. Use this table to understand which web applications and EAR files are updated when a product configuration is selected in the SAS Deployment Manager.

Product Configuration	EAR File
SAS Decision Services Design Middle Tier 6.4	(sas.decisionservices.designserver6.4.ear)
SAS Decision Services Engine Server 6.4	(sas.decisionservices.engine6.4.ear)
SAS Decision Services Monitor 6.4	(sas.decisionservices.monitor6.4.ear)

The web applications are rebuilt and redeployed with SAS Deployment Manager. For more information, see the *SAS Intelligence Platform: Middle-Tier Administration Guide*.

For more information about redeploying the SAS web applications, see the “Administering SAS Web Applications” chapter of the *SAS Intelligence Platform Middle-Tier Administrator’s Guide*.

Rebuilding the SAS Web Application Server Configurations

For more information about rebuilding the web application server configurations, see the “SAS Configuration Scripting Tools” chapter of *SAS Intelligence Platform Middle-Tier Administrator’s Guide*.

Post-Installation Reconfiguration

Engine and Design Server Reconfiguration

You must unconfigure and then reconfigure the SAS Decision Services engine and design servers, as well as the SAS Decision Services Monitor, using the SAS Deployment Wizard.

8

Migration

<i>Overview</i>	158
<i>Migration from SAS Real-Time Decision Manager 5.4</i>	159
<i>Migration from SAS Decision Services 5.5</i>	159
<i>Migration from SAS Decision Services 5.5M1</i>	160
<i>Migration from SAS Decision Services 5.6</i>	161
<i>Migration from SAS Decision Services 6.3</i>	162
<i>Migration from SAS Decision Services 6.4</i>	163
<i>Migrate Single DATA Step SAS Code</i>	163
<i>SAS Decision Services Utilities</i>	164
<i>Migrate Multiple DATA Step Code</i>	167
<i>Change Host Migration</i>	169
Federation, Database, and SAS Servers	169
<i>Password Updates</i>	171
<i>Migrate Monitor Database to Stand-alone PostgreSQL</i>	172

Overview

SAS Decision Services contains significant architectural advances over earlier releases. With these advances, some metadata and code objects were changed, some were replaced, and some were left unchanged. Here are some of the architectural changes, made after SAS Real-Time Decision Manager 5.4, that affect migration:

- SAS activity DS2 code is stored in the activity metadata, and is published when a flow that references the activity is activated.
- Design server now supports multiple repositories.
- The Scoring activity type has been removed. Score code is now published as a SAS activity.
- General I/O activity supports the additional operations: increment on update and upsert (insert if update fails).
- A high-performance endpoint interface has been added. Design-time batch simulations and the production of batch execution of flows are now supported.
- Monitoring has been enhanced to provide in-memory activity hit counts and performance statistics.
- A Restful administrative interface has been added.
- JGroups lateral cache has been replaced with VMware vFabric GemFire.

SAS Decision Services is shipped with DS2 versions of all activities.

Migration from the SAS Decision Services 6.3 to SAS Decision Services 6.4.

Before you upgrade, it is recommended that you export your flows, events, system resources, activities, and global variables manually before applying the upgrade. To export these resources:

- 1 Launch 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 Decision Services\SAS Decision Services 6.3`.
- 3** Right-click on the repository folder level, and select the export package menu item. Repeat this step for each folder repository.
- 4** Make a backup copy.

Migration from SAS Real-Time Decision Manager 5.4

To migrate from SAS Real-Time Decision Manager 5.4 to SAS Decision Services 6.4 manually, import your SAS Real-Time Decision Manager 5.4 metadata artifacts into the SAS Decision Services 6.4 repository.

Note: SAS Decision Services 6.4 does not support some of the older artifacts from SAS Real-Time Decision Manager 5.4. The import process updates the SAS Real-Time Decision Manager 5.4 objects for use in the SAS Decision Services 6.4 repository. However, multiple DATA step activities and SAS connection resources are no longer supported in SAS Decision Services 6.4. Therefore, the import process does not import these artifacts. As a result, the value of \$SAS_Activity_Resource is substituted in the activity XML for the system resource name in Activities that reference a SAS activity resource.

Migration from SAS Decision Services 5.5

To migrate from SAS Decision Services 5.5 to SAS Decision Services 6.4:

- 1** Install and configure the latest release of the SAS Federation Server, SAS Federation Server Manager, SAS Authentication Server, and Data Management

Studio. For more information, see *SAS Federation Server Administrator's Guide* and *SAS Authentication Server Administrator's Guide*.

- 2 Install SAS Decision Services 6.4. For more information, see *SAS Deployment Wizard and SAS Deployment Manager User's Guide*, available from the Install Center at <http://support.sas.com/documentation/installcenter/index.html>.
- 3 Previously published DS2 code must be modified and republished. Modify the PROC DS2 NOLIBS statement by referring to the example below. The required changes include removing SAS_ENCRYPT=YES and adding NOPROMPT and DRIVER=DS2. Republish the modified DS2 code to the new SAS Federation Server. Here is an example of the PROC DS2 NOLIBS statement.

```
proc ds2 nolibs noprompt="driver=remts;server=<federation server machine>;
port=<port>;protocol=bridge;uid=<federation server admin>;
pwd= <federation server admin password>';
conopts=(driver=ds2;CONOPTS=(DSN=BASE_DSN))";
```

Note: If you are upgrading a SAS Decision Services 5.5 deployment that was migrated from SAS Real-Time Decision Manager 5.4, the SAS Real-Time Decision Manager 5.4 single DATA step activities must be run through the SAS Decision Services 6.4 Migration utility in order to work properly in the SAS Decision Services 6.4 environment. For more information, see [“Migrate Single DATA Step SAS Code” on page 163](#).

Migration from SAS Decision Services 5.5M1

In order to migrate from the maintenance release for SAS Decision Services 5.5, install and configure the latest release of the SAS Federation Server, SAS Federation Server Manager, SAS Authentication Server, and Data Management Studio.

Note: The following resources are no longer used by SAS Decision Services and have been removed:

- SAS_Activity_Resource

- GeneralIO_Activity_Resource_FS
- \$Model_Update_Resource
- \$Score_JBCConnectionResource

Previously published DS2 code must be modified and republished. Modify the PROC DS2 NOLIBS statement by referring to the example below. The required changes include removing SAS_ENCRYPT=YES and adding NOPROMPT and DRIVER=DS2. Republish the modified DS2 code to the new SAS Federation Server. Here is an example of the PROC DS2 NOLIBS statement.

```
proc ds2 nolibs noprompt="driver=remts;server=<federation server machine>;
port=<port>;protocol=bridge;uid=<federation server admin>;
pwd= <federation server admin password>';
conopts=(driver=ds2;CONOPTS=(DSN=BASE_DSN)) ";
```

Note: If you are upgrading a SAS Decision Services 5.5M1 deployment that was migrated from SAS Real-Time Decision Manager 5.4, the SAS Real-Time Decision Manager 5.4 single DATA step activities must be run through the SAS Decision Services 6.4 Migration utility in order to work properly in the SAS Decision Services 6.4 environment. For more information, see [“Migrate Single DATA Step SAS Code” on page 163](#).

Migration from SAS Decision Services 5.6

In order to migrate from SAS Decision Services 5.6, install and configure the latest release of the SAS Federation Server, SAS Federation Server Manager, SAS Authentication Server, and Data Management Studio.

Note: The following resources are no longer used by Decision Services and have been removed:

- SAS_Activity_Resource
- GeneralIO_Activity_Resource_FS
- \$Model_Update_Resource

■ \$Score_JBCConnectionResource

Determine the artifacts that you want to migrate into the new design and production repositories and import the artifacts and their dependencies using the import facility in SAS Management Console.

The folder paths are renamed from `/System/Applications/SAS Decision Services/Decision Services 5.6` to `/System/Applications/SAS Decision Services/Decision Services 6.4`.

Only the SAS Deployment Wizard deployed design and production repositories are migrated.

Note: If you had published DS2 code in SAS Decision Services 5.5 or 5.5M1 and migrated to SAS Decision Services 5.6, then you must modify the DS2 code and republish it.

Modify the PROC DS2 NOLIBS statement by referring to the example below. The required changes include removing SAS_ENCRYPT=YES and adding NOPROMPT and DRIVER=DS2. Republish the modified DS2 code to the new SAS Federation Server. Here is an example of the PROC DS2 NOLIBS statement:

```
proc ds2 nolibs noprompt="driver=remts;server=<federation server machine>;
port=<port>;protocol=bridge;uid=<federation server admin>;
pwd= <federation server admin password>';
conopts=(driver=ds2;CONOPTS=(DSN=BASE_DSN))";
```

Migration from SAS Decision Services 6.3

In order to migrate from SAS Decision Services 6.3, determine the artifacts that you want to migrate into the new design and production repositories and import the artifacts and their dependencies using the import facility in SAS Management Console.

Only the SAS Deployment Wizard deployed design and production repositories are migrated.

For more information about migrating an existing deployment, see *About Using the SAS Migration Utility to Copy an Existing Deployment* at <http://support.sas.com/documentation/cdl/en/bisag/67481/HTML/default/viewer.htm#n0sb15gxhksdzln1vb8sxlcnrsf4.htm>.

Migration from SAS Decision Services 6.4

In order to migrate from SAS Decision Services 6.4, determine the artifacts that you want to migrate into the new design and production repositories and import the artifacts and their dependencies using the import facility in SAS Management Console.

Only the SAS Deployment Wizard deployed design and production repositories are migrated.

For more information about migrating an existing deployment, see *About Using the SAS Migration Utility to Copy an Existing Deployment* at <http://support.sas.com/documentation/cdl/en/bisag/67481/HTML/default/viewer.htm#n0sb15gxhksdzln1vb8sxlcnrsf4.htm>.

Migrate Single DATA Step SAS Code

The SAS code for SAS activities were DATA step fragments in SAS Real-Time Decision Manager 5.4 and earlier. The code must be migrated to use DS2 package methods in SAS Decision Services 6.3. You must manually execute the migrate script to migrate the SAS code for SAS activities. To migrate SAS activity code, issue the following from the command line:

Note: This is a single command.

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
6.3\Migrate -k
Enter activity filename here
C:/SAS/OneMachineRecommended/Lev1/Applications/
SASDecisionServicesServerConfig/Utilities/util.properties
```

SAS Decision Services Utilities

Various utilities are installed to assist with the migration of system resources and SAS activities. The SAS Decision Services utilities are located in `<SASHome>/SASDecisionServicesServerConfiguration/6.3/`.

Note: Execute the SAS Federation Server Migrate utility only if you created more SAS Federation Server configuration content (such as DSNs and data services) than what is required by SAS Decision Services 6.3.

The `Migrate` utility is provided to migrate SAS Decision Services SAS activities that were created in SAS Real-Time Decision Manager 5.4. In previous releases, these activities were DATA step fragments. In SAS Decision Services 6.3, the SAS activities are written as DS2 package methods. Here is the syntax for the `Migrate` utility.

```
Migrate [-k] [-f] [Activity File OR SASCode Directory]
        Property_Filename
```

Option	Description
-k	Keep the temporary files that are created as part of the migration process.
-f	Migrate files based on the <code>sc_programs.sas</code> file.
Activity File	The full path to the single DATA step SAS file to be migrated.
SASCode Directory	The full path to the SASCode directory containing <code>sas.exe.location</code> , <code>ds2.package.library</code> , and <code>ds2.migrate.dir</code> . This file is usually created during configuration and can be found at <code>SAS Config\Applications\SASDecisionServicesServerConfig\Utilities\util.properties</code> .

The UpdateResource utility is provided to easily change information that is stored in the SAS Decision Services Web Service Connection and JDBC Connection type system resources. Here is the syntax for the UpdateResource utility.

```
UpdateResource [-?] [-authdomain <current_value new_value>]
[-authpassword <current_value new_value>]
[-authuser <current_value new_value>] [-conopts <current_value new_value>]
[-debug] [-domain <domain>] [-driverclassname <current_value new_value>]
[-endpointaddress <current_value new_value>] [-host <hostname>]
[-jdbcpassword <current_value new_value>]
[-jdbcuser <current_value new_value>] [-log <log-file>] [-nolog]
[-password <password>] [-port <port>] [-profile <profile>]
[-proxyhost <current_value new_value>]
[-proxypassword <current_value new_value>]
[-proxyport <current_value new_value>]
[-proxyuser <current_value new_value>]
[-repository <repository>] [-resource <resource>]
[-serverurl <current_value new_value>] [-user <userid>]
[-wsdluri <current_value new_value>]
```

Option	Description
-?,-help	Prints help information.
-debug	Prints debugging information.
-log <log-file>	Log file or directory.
-nolog	Disable log file.
-repository <repository>	The repository name.
-resource <resource>	The resource type.

Note: The metadata server connection options are required if -profile is not set or if the profile does not contain connection credentials.

Metadata Server Connection Option	Description
-port <port>	Metadata server port.
-domain <domain>	User authentication domain.

-host <hostname>	Metadata server host.
-password <password>	User's login password.
-user <userid>	User's login identity.
-profile	Can be used in place of the -host, -port, -user, and -password options.

Web Service Connection Resource Authentication Option

Description

-authdomain <current_value new_value>	The domain value.
-authpassword <current_value new_value>	The password value.
-authuser <current_value new_value>	The user value.
-proxyhost <current_value new_value>	The proxy host value.
-proxypassword <current_value new_value>	The proxy password value.
-proxyport <current_value new_value>	The proxy port value.
-proxyuser <current_value new_value>	The proxy user value.
-wsdluri <current_value new_value>	The WSDL URI value.

JDBC Option

Description

-driverclassname <current_value new_value>	The driver class name value.
-endpointaddress <current_value new_value>	The endpoint address value.

JDBC Connection Resource Option

Description

-jdbcpassword <current_value new_value>	The password value.
-jdbcuser <current_value new_value>	The user value.

<code>-serverurl <current_value new_value></code>	The server URL value.
ConOpts Option	Description
<code>-conopts <current_value new_value></code>	The ConOpts value.

Once you have run the UpdateResource utility, the following items have been migrated:

- 1 The repositories SASDSDesignRepository and SASDSEngineRepository and their contents are migrated automatically using the migration utility.
- 2 The SAS Real-Time Decision Server Config software component is migrated to the metadata folder `/System/Application/SAS Decision Services/Decision Services 6.3` and renamed to Decision Services Server Config 6.3.

Migrate Multiple DATA Step Code

Note: This information pertains only to migrations that involve SAS Real-Time Decision Manager 5.4.

Multiple DATA step code must be migrated manually. Skip these steps if your company, or your on-site SAS personnel, has not created new custom multiple DATA step activities. Otherwise, perform the following steps to migrate your multiple DATA step code to a stored process:

- 1 Convert the SAS code to a stored process that uses input and output parameters as macro variables. In most cases, you remove the `%macro` and `%mend` statements from your multiple DATA step SAS code. For code that uses tables and arrays, references to `%sc_encode_decode` must be changed to `%scencode`.
- 2 Load the stored process into the metadata. If WIP is running, it is available as a SAS BI web service.

- 3 Modify consuming flows to replace all references of the multiple DATA step SAS activity with the SAS BI web service activity.
- 4 Connect the input and output parameters of the flow to the BI web services using XPath expressions. If you need help with XPath expressions, or with any of the other steps, contact your on-site SAS support personnel, or SAS Technical Support.
- 5 Open SAS Management Console. Select the metadata profile of a user who has permissions on the design and production repositories.
- 6 From the **Folder** tab, navigate to **System ► Applications ► SAS Decision Services ► SAS Decision Services 5.5 ► SASDSEngineRepository**.
- 7 Right-click **SASDSEngineRepository**, and click **New System Resource**.
- 8 From the drop-down menu, select **Web Service**.
- 9 Create a web service resource that points to the URL of the BI web service (for example, `http://host:port/SASBIWS/services/Shared Data/MyStoredprocesses/myStoredProcess`).
- 10 Click **OK**.
- 11 Repeat the steps for **SASDSDesignRepository**.
- 12 In both the design and engine repositories, create a web service activity that points to the web service resource.

See the *SAS BI Web Services Developer's Guide*, available at <http://support.sas.com/documentation/cdl/en/wbsvcdg/64883/PDF/default/wbsvcdg.pdf>, for more information.

Change Host Migration

Federation, Database, and SAS Servers

SAS Federation Server and Database Server Manual Migration

Because SAS Decision Services can be configured for access to any schema or table in your database, automatic database server content migration is not possible. SAS Federation Server and SAS Authentication Server content migration is a manual process.

If the database server was migrated to a different host, then load the DS2 packages into the new database server.

If the SAS Federation Server and SAS Authentication Server are migrated to a different host, then you must manually reconfigure the DSNs, data services, users, groups, and base schema and catalog.

JDBC Connection Resources

If the database server host or the SAS Federation Server were migrated, then change the server URL for each JDBC Connection resource. You can execute the following commands that perform this task for each JDBC Connection resource. To change the server host name, perform the following steps:

- 1 Issue the following from the command line:

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
  6.3\UpdateResource
-host Host location -port Port number -user Enter username -password
Enter password -repository SASDSEngineRepository -resource
JDBCConnectionResource -serverurl current_value new_value
```

- 2 Issue the following from the command line:

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
  6.3\UpdateResource
-host Host location -port Port number -user Enter username -password
Enter password -repository SASDSDesignRepository -resource
```

```
JDBCConnectionResource -serverurl current_value new_value
```

If the database server host or the SAS Federation Server were migrated, then you must change the connection options for the SAS Federation Server JDBC Connection resources. You can execute commands that perform this task for each JDBC Connection resource. To change the connection options, perform the following steps:

1 Issue the following from the command line:

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
  6.3\UpdateResource
-host Host location -port Port number -user Enter username -password
Enter password -repository SASDSEngineRepository -resource
JDBCConnectionResource -conopts current_value new_value
```

2 Issue the following from the command line:

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
  6.3\UpdateResource
-host Host location -port Port number -user Enter username -password
Enter password -repository SASDSDesignRepository -resource
JDBCConnectionResource -conopts current_value new_value
```

Web Service Resources

If the WIP server was migrated, then change the WSDL URL for each web service resource. You can create a batch file that performs this task for each web service resource. To change the endpoint address:

1 Issue the following from the command line:

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
  6.3\UpdateResource
-host Host location -port Port number -user Enter username -password
Enter password -repository SASDSEngineRepository -resource
WSConnectionResource -wsdluri current_value new_value
```

2 Issue the following from the command line:

```
C:\Program Files\SASHome\SASDecisionServicesServerConfiguration\
  6.3\UpdateResource
-host Host location -port Port number -user Enter username -password
Enter password -repository SASDSDesignRepository -resource
WSConnectionResource -wsdluri current_value new_value
```

- 3 Redeploy the BI web services for your SAS Decision Services stored processes.

Password Updates

The following table summarizes details about the passwords that were updated by the SAS Deployment Manager and the passwords that must be updated manually.

The path to the resources in the SAS Metadata Repository is `SAS Folders\System\Applications\SAS Decision Services\Decision Services 6.4\<repository folder>`. There is usually a design repository folder and an engine repository folder. However, that might not always be the case.

Note: Passwords that are controlled by an external provider (such as in LDAP, Active Directory, or the host operating system) are not synchronized. You must ensure that the passwords that you provide as input match the actual passwords in your external provider.

User ID	Usage	Update Description
SAS Spawned Server Account	Used by activity publishing.	This is updated automatically when Update Passwords is selected from SAS Deployment Manager.
SAS Federation Server system user ID	Used in the SAS Decision Services system resource.	Update the Decision Services \$SAS_Activity_Resource using SAS Management Console or the Decision Services UpdateResource utility. For more information about the UpdateResource utility, see the “SAS Decision Services Utilities” on page 164.

User ID	Usage	Update Description
Decision Services Web Infrastructure Platform Data Server user account	Used to store monitor, user, and batch tables. Used in the SAS Decision Services system resource.	<p>This is updated automatically in the SAS_Server_7 application server data source and the Web Infrastructure Platform Data Server when Update Passwords is selected from the SAS Deployment Manager.</p> <p>Update \$User_Log_ JDBCConnectionResource using SAS Management Console or the SAS Decision Services UpdateResource utility. For more information about the UpdateResource utility, see the “SAS Decision Services Utilities” on page 164.</p>
Customer Data Access Database user ID	Used in the Decision Services system resource.	<p>Update the SAS Decision Services GeneralIO_Activity_Resource using SAS Management Console or the SAS Decision Services UpdateResource utility. For more information about the UpdateResource utility, see the “SAS Decision Services Utilities” on page 164.</p>

Migrate Monitor Database to Stand-alone PostgreSQL

You must migrate the Decision Services Monitoring data to a stand-alone postgres database if the performance of the default SAS Web Infrastructure Platform Data Server

degrades and affects the SharedServices database. There are various methods for moving data to a new PostgreSQL server. Here is an example of one method. The `pg_dump` and `psql` commands can be executed on the source SAS Web Infrastructure Platform Data Server or the PostgreSQL server. These instructions assume that the commands are executed from the target server. On UNIX, you might need to create the DecisionServices user.

- 1 Stop the application server where the Decision Services Monitor is located (such as SASServer7_1).
- 2 Locate the file *SAS Config Dir/Lev1/Web/WebAppServer/SASServer7_1/conf/server.xml*.

- 3 In the server.xml file, locate this resource:

```
<Resource auth="Container" driverClassName="org.postgresql.Driver"
    factory="com.atomikos.tomcat.NonXABeanFactory" maxPoolSize="100"
    minPoolSize="10" name="sas/jdbc/DecisionServices"
    password="{pw.sas.jdbc.DecisionServices}" testQuery="select 1"
    type="com.atomikos.jdbc.nonxa.AtomikosNonXADataSourceBean"
    uniqueResourceName="sas/jdbc/DecisionServices"
    url="jdbc:postgresql://hostname.domain.com:9432/DecisionServices"
    user="DecisionServices"/>
```

Then, modify the host name to point to the host and port of the stand-alone PostgreSQL server.

- 4 On the Decision Services Monitor server, locate the file *SAS Config Dir/Lev1/Web/Applications\SASDecisionServicesMonitor6.3/create-dcsv-data-standalone1.sql* and copy it to a location on the target postgres server. Edit this file to modify the password for the DecisionServices user.
- 5 Add the path *Postgres Install Dir\PostgreSQL\9.4\bin* to your path.
- 6 On the target PostgreSQL server, execute this command:

```
pg_dump -h WIP data server hostname -p
WIP data server port -n monitor -U
DecisionServices DecisionServices > dcsvc_decisionservices.sql
```

- 7 On the target PostgreSQL server, execute these commands:

```
psql -h localhost -p 5432 -U  
postgres admin user < create-dcsv-data-standalone.sql
```

```
b.    psql -h localhost -p 5432 -U  
DecisionServices < dcsvc_decisionservices.sql
```

- 8** Start the application server where the Decision Services Monitor is located (such as SASServer7_1).

9

Best Practices

<i>SAS Server Options</i>	176
<i>JDBC Performance Tuning</i>	176
Tuning Controls	176
Tuning Considerations for SAS Federation Servers	181
Tuning Considerations for Database Management Systems ...	182
Keeping Connections Alive	182
<i>HTTP Client Code Usage</i>	183
Overview	183
SAS Customer Experience Real-Time Server Integration	184
Web Service Integration	185
SAS Decision Services Client API for Java	185
<i>Use of SAS Data Sets</i>	186
<i>Initialize DS2 Activity Variables</i>	187
<i>Using SQLSTMT in Real-Time Applications</i>	187
Overview	187
Example 1: SQL Select Statement with Bound Variables	188
Example 2: SQL Select Statement without a Bound Variable ...	190
Example 3: SQL Insert Statement Using Bound Parameters ...	191
<i>Suggested Initial Values for Key Tuning</i>	
<i>Elements in Systems with Heavy Workloads</i>	193
Overview	193

Best Practices for Optimum Performance	195
<i>Incorporating Pool Diagnostics</i>	196
Overview	196
Accessing the PoolDiagnostics.jsp Page	197
Items Displayed for Activity and Sub Flow Thread Pool	197
Items Displayed for JDBC Connection Pools	198

SAS Server Options

Some options can be edited to improve the performance of SAS Federation 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 significantly reduces overall system performance. The logging levels that affect your DS2 activities are specified by the SAS logging facility configuration file that is installed with your SAS Federation Server.

By default, the logging level is set to `Info`. For production, the application logger should be changed to `Error`.

```
<!-- Application message logger -->
<logger name="App">
  <level value="Info"/>
</logger>
```

After you disable logging, you must restart SAS Federation Server.

JDBC Performance Tuning

Tuning Controls

Two JDBC Connection resources are configured for three separate purposes:

- To allow decision service activities to execute DS2 package methods, which are hosted by SAS Federation Server. These package methods are used for scoring, or for executing custom SAS code.

- To enable General I/O activities to read from and write to relational databases such as Oracle, DB2, Teradata, and Microsoft SQL Server.
- To enable General I/O activities, which are configured to access SAS data sets, to read SAS data sets through SAS Federation Driver for Base SAS.

SAS Decision Services uses Apache Commons Database Connection Pooling (DBCP) to affect efficient caching and management of JDBC Connections, parameterized prepared statements, and parameterized callable statements. For more information about Apache Commons DBCP, see <http://commons.apache.org/dbcp/index.html>.

DBCP pool tuning values are stored in JDBC Connection resources. To access the pool tuning controls, select the **Folders** tab of SAS Management Console, and follow these steps:

- 1 On the **Folders** tab, expand **System ► Applications ► SAS Decision Services ► Decision Services 6.4**.
- 2 Right-click the JDBC Connection resource that you want to configure, and select **Edit System Resource**.

Additional JDBC Connection resources can be added to enable access to additional database management systems.

Note: The number of free database connections that are needed by the SAS Decision Services engine can be estimated by adding up the maximum number of connections for each JDBC Connection resource that connects to the database in the Decision Services Manager plug-in.

Figure 9.1 Create a JDBC System Resource - Advanced

Create New "JDBC Connection" Resource

Name:

Description:

Driver Manager Setting

Driver Class:

Server URL:

Connection Options:

User Name:

Password:

Connection Pooling

Configured	Name	Value
<input type="checkbox"/>	Lifo	false
<input type="checkbox"/>	MaxActive	
<input type="checkbox"/>	MaxIdle	
<input type="checkbox"/>	MaxWait	
<input type="checkbox"/>	MaxTotal	
<input type="checkbox"/>	MinEvictableIdleTimeMillis	
<input type="checkbox"/>	MinIdle	

Statement Pooling

Configured	Name	Value
<input type="checkbox"/>	Lifo	false
<input type="checkbox"/>	MaxActive	
<input type="checkbox"/>	MaxIdle	
<input type="checkbox"/>	MaxWait	
<input type="checkbox"/>	MaxTotal	
<input type="checkbox"/>	MinEvictableIdleTimeMillis	
<input type="checkbox"/>	MinIdle	

Hide Advanced

OK

Cancel

Help

Each attribute can be disabled or enabled by selecting the check box in the Configured column. You can also enable or disable all attributes by selecting **Enable All**. If all of the

attributes are disabled in either Connection Pooling or Statement Pooling, the XML element is not created in the JDBC resource. If the pooling control is saved with the JDBC resource, you will see the advanced dialog box when you edit this system resource the next time. You can click **Reset to Default** to return to the basic dialog box. Any connection or statement pool setting that is not explicitly configured in the JDBC resource, uses a default value that is defined by Apache Commons DBCP. These default values are listed below in the descriptions of each pool tuning control.

After you click **OK**, the new JDBC Connection system resource appears in the repository.

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

Lifo

determines whether the pool returns idle objects in last-in-first-out order. The default setting for this parameter is `true`.

MaxActive

controls the maximum number of objects that can be allocated by the pool (checked out to clients or idle awaiting check-out) at a given time. When the value is non-positive, there is no limit to the number of objects that can be managed by the pool at one time. When MaxActive is reached, the pool is said to be exhausted. The default setting for this parameter is 8.

MaxIdle

controls the maximum number of objects that can sit idle in the pool at any time. When the value is negative, there is no limit to the number of objects that can be idle at one time. The default setting for this parameter is 8.

MaxWait

the maximum amount of time, in milliseconds, to wait for an idle object when the pool is exhausted. The default setting for this parameter is -1, which means the wait can continue indefinitely.

MaxTotal

sets a global limit on the number of objects that can be in circulation (active or idle) within the combined set of pools. When the value is non-positive, there is no limit to the total number of objects in circulation. When maxTotal is exceeded, all keyed pools are exhausted. When maxTotal is set to a positive value, and borrowObject is

invoked when at the limit with no idle instances available, an attempt is made to create room by clearing the oldest 15% of the elements from the keyed pools. The default setting for this parameter is -1 (no limit).

MinEvictableIdleTimeMillis

specifies the minimum amount of time that an object can sit idle in the pool before it is eligible for eviction because of idle time. When the value is non-positive, no object is dropped from the pool because of idle time alone. This setting has no effect unless `TimeBetweenEvictionRunsMillis` is greater than 0. The default setting for this parameter is 30 minutes.

MinIdle

sets a target value for the minimum number of idle objects (per key) that should always be available. If this parameter is set to a positive number and `timeBetweenEvictionRunsMillis` is greater than 0, each time the idle object eviction thread runs, it tries to create enough idle instances so that the specified number of idle instances will be available under each key. This parameter is also used by `preparePool`, if `true` is provided as that method's `populateImmediately` parameter. The default setting for this parameter is 0.

NumTestsPerEvictionRun

determines the number of objects to be examined in each run of the idle object evictor. This setting has no effect unless `TimeBetweenEvictionRunsMillis` is greater than 0. The default setting for this parameter is 3.

TestOnBorrow

whether to validate objects before they are returned by the `borrowObject()` method.

TestOnReturn

whether to validate objects after they are returned to the `returnObject(java.lang.Object)` method.

TestWhileIdle

whether to validate objects in the idle object eviction thread, if any.

TimeBetweenEvictionRunsMillis

the amount of time (in milliseconds) to sleep between examining idle objects for eviction.

WhenExhaustedAction

specifies the behavior of the `borrowObject()` method when the pool is exhausted.

SoftMinEvictableIdleTimeMillis

the minimum number of milliseconds an object can sit idle in the pool before it is eligible for eviction. There is an extra condition that at least `MinIdle` number of objects remain in the pool.

Tuning Considerations for SAS Federation Servers

A SAS Federation Server maintains a thread of execution per open JDBC Connection. Therefore, the size of the connection pool has a direct effect on the number of threads that are available to service requests for activity method execution. For best performance, you want SAS Federation Server to maintain an optimum number of ready-to-run threads. For this reason, `maxIdle` and `maxActive` should be set to the same value, so that idle connections are not closed, and you want this value to equal the optimum number of threads. Because of the wide variation in server capabilities, you might need to experiment to find this optimum number. A good starting point is to set `maxIdle` and `maxActive` equal to twice the number of cores in SAS Federation Server. Adjust this number up and down while measuring CPU use, latency, and throughput in order to achieve an optimal setting. Specifying too few threads under uses the hardware, and too many threads causes SAS Federation Server to thrash. Therefore, these settings are critical to the performance of your system.

The size of the statement pool should be large enough to contain an entry for every activity method deployed to the system. A statement pool is allocated per connection, so do not multiply the number of statements by the number of connections. Instead, simply use the number of statements as the `maxActive` value. If memory is at a premium, the `maxIdle` value can be adjusted down to reclaim space taken up by methods that are only rarely called.

Tuning Considerations for Database Management Systems

Database performance tuning is a highly complex and specialized topic beyond the scope of this document. It depends on many factors, including network bandwidth, cache size, data transfer rates, disk array configuration, application characteristics, and more. See your database management system vendor for assistance.

Keeping Connections Alive

A database connection is essentially a socket connection. Operating systems, database hosts, and firewalls drop idle TCP connections after a period of time. The following settings, which are not default, run an evictor thread every 30 minutes to evict any idle connections older than 30 minutes. This prevents connection failures because of invalid TCP connections, in most cases. The connection pools are reset when the system is synchronized. Synchronization occurs whenever a flow is activated or deactivated, or when **Synchronize with Repository** is selected from the Decision Services Manager plug-in for SAS Management Console.

Setting	Example
TimeBetweenEvictionRunsMillis	1000 * 60 * 30
NumTestsPerEvictionRun	3
MinEvictableIdleTimeMillis	1000 * 60 * 30

There is a one-to-one relationship between the number of request processing threads that are allocated within SAS Federation Server and the size of the SAS Federation Server JDBC connection pool. Therefore, pool size affects performance.

See your database vendor for information about performance tuning JDBC connection and statement pools.

HTTP Client Code Usage

Overview

SAS Decision Services uses Apache HTTP Client code in the following areas:

- SAS Customer Experience Real-Time Server integration
- Web service integration (including the SAS Customer Intelligence Contact and Response History)
- SAS Decision Services client API for Java

The Apache HTTP Client exposes a number of configuration parameters through its preferences API: <http://hc.apache.org/httpclient-3.x/preference-api.html>. Choose parameter values carefully to ensure good performance. Here are common values:

`http.protocol.version`

Set this value to HTTP/1.1, as it is more efficient.

`http.protocol.expect-continue`

Set this value to false for SAS Customer Experience Real-Time Server integration, SAS Decision Services client API for Java, and for SAS Customer Intelligence Common Services. It eliminates the step where the client determines whether the server is willing to accept the request. In most cases, the server is willing.

`http.protocol.cookie-policy`

Set this value to ignore cookies. The use cases do not require them.

`http.socket.timeout`

In most cases, set this value to 1000 ms. Often, SAS Decision Services must support a subsecond response. A different value can be set depending on your performance requirements. A large time-out value can result in a less responsive system.

`http.tcp.nodelay`

A value of true is recommended because it reduces network latency.

`http.connection.timeout`

In most cases, set this value to 1000 ms. This is the time to create a new connection. Because connections are pooled at steady state, new connections are not created as often.

`http.connection.stalecheck`

A value of false is recommended so that the success case is faster. Setting this value to true causes the client to check every connection before it is used, which adds time to every call.

`http-connection-manager.max-per-host`

Set this value based on server capability. For example, if the server can support 1000 concurrent HTTP connections, set this value at 1000. Every instance of the HTTP Connection system resource, web service activity resource, and the SAS Decision Service RequestFactory creates a pool of connections, all for the same URL (for example, the same host).

`http-connection-manager.max-total`

Set this value to be the same as max-per-host because every pool supports only a single host.

`http-connection-manager.class`

Set this value to
`org.apache.commons.httpclient.MultiThreadedHttpConnectionManager`.

SAS Customer Experience Real-Time Server Integration

For SAS Customer Experience Real-Time Server integration, the parameters can be set by editing the HTTP Connection system resource in the SAS Decision Services plug-in for SAS Management Console. For more information, see [“Specify a New System Resource as an HTTP Connection” on page 77](#).

Web Service Integration

Not all of the Apache HTTP Client properties are supported. The following properties are supported and are set using Java system properties, which can affect all web service activities and resource combinations.

System Properties	HTTP Client Properties	Default Values (if not set)
com.sas.sasds.maxHostConnections	http-connection-manager.max-per-host	1000
com.sas.sasds.maxTotalConnections	http-connection-manager.max-total	10000
com.sas.sasds.staleChecking Enabled	http.connection.stalecheck	true
com.sas.sasds.tcpNoDelay	http.tcp.nodelay	true
com.sas.sasds.connectionTimeout	http.connection.timeout	1000

SAS Decision Services Client API for Java

The properties that are mentioned above can be set by passing a Java properties object that contains the name and value of the parameters when instantiating the SASDSRequestFactory object. Here is an example:

```
String uri = "http://abcd.com/test";
Properties props = new Properties();

props.set("http.protocol.version", "HTTP/1.1");
props.set("http.connection.timeout", "1000");
props.set("http.tcp.nodelay", "true");

SASDSRequestFactory factory = SASDSRequestFactory.getInstance(uri, props);
```

Note: These values are always set as String values.

Use of SAS Data Sets

SAS data sets can be used by SAS Decision Services to hold data, as well as DS2 activity code and score code. To access data that is held in SAS data sets, you must configure a DSN in SAS Federation Server that uses the Base driver, and then use this DSN (or a federated DSN that includes this DSN) in the resource definition.

The folder that contains the data sets must be accessible to SAS Federation Server. If multiple SAS Federation Servers are in use to implement failover or load balancing, then the data sets must be held in a shared folder that is accessible to all SAS Federation Servers. This typically requires SAS Federation Server to have Read access to this shared folder.

Because SAS data sets do not support concurrent access for writing to them, they are commonly used as Read-only.

It is possible to write to SAS data sets through the SAS Federation Server by disabling concurrent access. However, this practice is not recommended because of the large performance penalty that it carries. This is done by limiting the topology to include a single SAS Federation Server and a single engine node, and then setting the size of the connection pool for the resource to 1. Reducing the number of connections reduces the number of threads for this connection and affects throughput for this resource. For applications that must write data, it is strongly recommended to use a relational database that supports concurrent writes. This includes usages like user logging, as well as reading and writing business data.

SAS data sets can also be used to hold DS2 activity code and score code. These are read by SAS Federation Server when they are first referenced by a decision flow. As mentioned earlier, for configurations using multiple SAS Federation Servers, the data sets must be held in a shared folder accessible by all SAS Federation Servers.

Initialize DS2 Activity Variables

When you are programming in DS2, unassigned variables are not initialized to missing as they are in a DATA Step. As a best practice, you should explicitly initialize all DS2 variables. Related variables with package scope retain their values across method invocations, unless they are explicitly reinitialized. A package scope variable is a variable that is declared prior to the first method declaration of a DS2 package.

Using SQLSTMT in Real-Time Applications

Overview

This section describes how best to take advantage of the SQLSTMT package, which first became available with SAS Federation Server 3.2. In addition to the SQLSTMT package, several other noteworthy performance improvements became available with SAS Federation Server 4.1:

- Memory management improvements
- Greater stability, as code base closely matches the SAS 9.4 DS2 code base
- Access to MDS, an in-memory data store

It is advantageous to use SQLSTMT rather than the hash object, for the following reason:

- SQLSTMT allows a query to be prepared once and executed many times. The hash object must prepare the query each time it is executed.
- SQLSTMT binds parameters, which is essential to quick execution.
- SQLSTMT supports inserts, updates, selects, and other complex SQL operations.

To use the SQLSTMT package, perform the following steps:

- 1 Declare the package instance as a package-level variable (before the first method declaration of your DS2 package).
- 2 Execute the package from a method.
- 3 In cases of select statements, fetch the returned data.

Because you used a package-level variable to hold the SQLSTMT instance, it is not destroyed when your method returns.

Example 1: SQL Select Statement with Bound Variables

This example assumes the existence of a catalog called oraSource, with a schema called mySchema, and a table called testable. The table contains three columns: myInt of type bigint, myString of type varchar, and myFloat of type double. This example binds the input parameter and the result set values to package variables. It should be noted that, when binding a variable to an SQLSTMT package, a package-level variable must be used. This is a best practice, as it avoids making an extra copy of the variables, unlike when using the SET and Get method, which is described below. Binding becomes especially important where large string variables are involved. See the inline comments for more details.

```
package example1 /overwrite=yes;
/* Note: All variables used as bound parameter variables must be package
   level variables. */

/* A package level variable is used to hold the customer ID that is bound
   to the select statement. */
dcl bigint _custId;

/* Package level variables are used to bind the result set data. */
dcl bigint _myInt;
dcl varchar(255) _myString;
dcl double _myFloat;

/* A package level variable that gets prepared once when the package instance
   is created. */
dcl package SQLSTMT _selectData('SELECT myInt, myString, myFloat FROM
oraSource.mySchema.testTable WHERE custId=?', [_custId]);
```

```

    /* An execute method is called from the middle tier. */
    method execute(bigint customerID);
/* An integer variable is used to hold the return code from the function
   executions. */
    dcl int rc;

/* Set the bound parameter to the value of customer ID passed in from the
   middle tier. */
    _custId = customerID;

    /* Execute the select statement. */
    rc = _selectData.execute();

    /* If rc equals 1, then there was an error executing. */
    /* This should be logged and the method should return. */
    if (rc = 1) then do;
        /* Log an error message. */
        return;
    end;

    /* Bind the result set columns. */
    rc = _selectData.bindresults([_myInt, _myString, _myFloat]);

    /* If rc equals 1, then there was an error binding data. */
    /* This should be logged and the method should return. */
    if (rc = 1) then do;
        /* Log an error message. */
        return;
    end;

    /* Fetch the first row from the result set. */
    rc = _selectData.fetch();

    /* If rc equals 1, then there was an error fetching data. */
    /* This should be logged and the method should return. */
    if (rc = 1) then do;
        /* Log an error message. */
        return;
    end;

    /* A return code of 2 indicates that there is no more data. */
    /* In the result set, while there is data, process it */
    do while(rc ~= 2);
        /* Do something with the data. */

        /* Fetch the next row from the result set. */

```

```

        rc = _selectData.fetch();
    end;
end;
endpackage;
run;

```

Example 2: SQL Select Statement without a Bound Variable

This example assumes the existence of a catalog called oraSource, with a schema called mySchema, and a table called testable. The table contains three columns: myInt of type bigint, myString of type varchar, and myFloat of type double. In this example, set and get methods are used to access parameters and result set data. See the inline comments for more details.

```

package example2 /overwrite=yes;

/* A package level variable that gets prepared once when the package instance
   is created. */
    dcl package SQLSTMT _selectData('SELECT myInt, myString, myFloat FROM
        oraSource.mySchema.testTable WHERE custId=?');

    /* Execute the method called from the middle tier. */
    method execute(bigint customerID);
/* An integer variable is used to hold the return code from function executions. */
    dcl int rc;

    /* Local variables to hold data from a single row of the result set. */
    dcl bigint myInt;
    dcl varchar(255) myString;
    dcl double myFloat;

/* Set the parameter to the value of customer ID passed in from the middle tier. */
    _selectData.SETBIGINT(1, customerID);

    /* Execute the select statement. */
    rc = _selectData.execute();

    /* If rc equals 1, then there was an error executing. */
    /* This should be logged and the method should return. */
    if (rc = 1) then do;
        /* Log an error message. */
        return;
    end;

```



```

/* Fetch the first row from the result set. */
rc = _selectData.fetch();

/* If rc equals 1, then there was an error executing. */
/* This should be logged and the method should return.      */
if (rc = 1) then do;
    /* Log an error message. */
    return;
end;

/* A return code of 2 indicates that there is no more data. */
/* In the result set, while there is data, process it.      */
do while(rc ~= 2);
    myInt = _selectData.GETBIGINT(1);
    myString = _selectData.GETVARCHAR(2);
    myFloat = _selectData.GETDOUBLE(3);
    /* Do something with the data. */

    /* Fetch the next row from the result set. */
    rc = _selectData.fetch();
end;

end;
endpackage;
run;

```

Example 3: SQL Insert Statement Using Bound Parameters

This example assumes the existence of a catalog called oraSource, with a schema called mySchema, and a table called testable. The table contains three columns: myInt of type bigint, myString of type varchar, and myFloat of type double. A single row of data is inserted based on the value passed in from the middle tier. See the inline comments for more details.

```

package example3 /overwrite=yes;
/* Note: All variables used as bound parameter variables must be package
   level variables. */

/* A package level variable used to hold the customer ID that is bound
   to the select statement. */
dcl bigint _custId;

/* Package level variables used to bind the result set data */
dcl bigint _myInt;

```

```

    dcl varchar(255) _myString;
    dcl double _myFloat;

/* A package level variable that gets prepared once when package
instance is created. */
    dcl package SQLSTMT _insertData('INSERT INTO oraSource.mySchema.testTable
SET custId=?, myInt=?, myString=?, myFloat=?',
[_custId, _myInt, _myString, _myFloat]);

/* An execute method is called from the middle tier. */
method execute(bigint customerID, bigint myInt, varchar(255) myString,
double myFloat);
/* An integer variable is used to hold the return code from function executions. */
    dcl int rc;

/* Set the bound parameters to the values passed in from the middle tier. */
    _custId = customerID;
    _myInt = myInt;
    _myString = myString;
    _myFloat = myFloat;

/* Execute the insert statement */
    rc = _insertData.execute();

/* If rc equals 1, then there was an error executing. */
/* This should be logged and the method should return. */
    if (rc = 1) then do;
        /* Log an error message. */
        return;
    end;
end;
endpackage;
run;

```

Suggested Initial Values for Key Tuning Elements in Systems with Heavy Workloads

Overview

SAS Real-Time Decision Manager treatment and treatment set campaigns are the most resource intensive applications run by SAS Decision Services. These applications require comprehensive tuning of all components to achieve best performance. The settings found in this section represent reasonable initial values for treatment set applications. The settings are not necessarily appropriate for other, less resource intensive, applications.

The reference application, to which the following initial tuning settings apply, includes 23 concurrent treatment campaigns in a single treatment set. Each treatment campaign calls custom DS2 code, which in turn queries an Oracle database up to six times per transaction.

Note: These settings are only a suggested starting point. Additional tuning should be conducted using your specific application on your specific hardware, while simulating peak transaction volumes. The Pool Diagnostics JSP, described in [“Incorporating Pool Diagnostics” on page 196](#), is a useful tool for measuring the effects of your tuning adjustments.

Table 9.1 Reference System

Operating system	AIX 6.1
SAS Customer Intelligence or SAS Real-Time Decision Manager version	6.3
SAS Decision Services version	6.3
SAS version	9.4M1

SAS Federation Server version	4.1
Number of engine servers (physical)	2
Number of SAS Federation Servers (physical)	2
Are engine and SAS Federation Server collocated or dislocated?	Dislocated, although it is recommended that the engine and SAS Federation Server be collocated.
Processor	IBM Power 7
Cores per engine server (available/used)	12/4-5
Cores per SAS Federation Server (available/used)	12/1-2
RAM per compute server	32GB

Table 9.2 Application

Type	SAS Real-Time Decision Manager treatment sets with sort arbitration
Number of concurrent treatment campaigns	23
Number of DS2 custom packages	10
Number of I/O calls to Oracle from DS2	6 (several per transaction)

Table 9.3 Key Turning Parameter Values

Engine JVM memory (setting/used)	16384/12288 MB
SAS Federation Server instances per compute server	1
MaxActive (SAS_Activity_Resource)	5
MaxIdle(SAS_Activity_Resource)	5

MinIdle(SAS_Activity_Resource)	5
Total number of TK worker threads	20 (5x1x2x2)
Activity execution thread pool size	1600

Table 9.4 *Observed Performance*

Average transaction response time	0.5 seconds
Transaction latency (service level agreement)	1-2 seconds
Observed throughput	73 tps
Peak transaction load (service level agreement)	16 tps

Best Practices for Optimum Performance

The following is a list of methods to optimize performance:

- Make use of the SQLSTMT package. SQLSTMT allows the use of bind variables and package level SQLSTMT declarations, which dramatically improves speed and resource utilization.

Note: The SQLSTMT packages use considerable more memory than hash objects because the statements are cached for each JDBC connection. However, this memory utilization is a good trade-off for performance.
- Run one SAS Federation Server per host. SAS Federation Server's improved internal memory management reduces threading contention and eliminates the need to run more than one instance per host.
- Minimize use of the text concatenation operator, where possible, in DS2. This operator (||) causes excessive CPU on the SAS Federation Server.
- Increase the number of threads assigned to the activity.execution.thread.pool from the default of 100 to 600.

- Perform as much work in the database as possible. Avoid joining tables at run time.
- Always index your database tables.

The following tasks are critical to achieving best performance:

- Set the appropriate number of JDBC connections to the SAS Federation Server.
- Set the appropriate number of threads available to the execution thread pool.
- Write DS2 code with real-time performance in mind, using the SQLSTMT with package level declarations and bind variables, rather than the DS2 hash object.

To set the first two values appropriately, you must understand the number of activities that are being executed in each decision flow and the volume of concurrent requests.

Incorporating Pool Diagnostics

Overview

SAS Decision Services uses resources like threads and database connections that are pooled and whose pools need to be configured optimally for the best throughput. PoolDiagnostics.jsp is a web page on the SAS Decision Services engine web application that displays the status of the primary thread pool and the JDBC connection pools for that engine. Based on the data displayed, the pool configuration can be adjusted for the best performance.

There are two types of pools that the PoolDiagnostics.jsp can address:

Activity and sub flow thread pool

Also known as the primary thread pool. The SAS Decision Service engine is a web application that executes decision flows that call out to other subflows and activities. While the incoming request is executed in the application server's servlet thread pool, the calls to sub flows and activities are executed on threads borrowed from the primary thread pool. If not enough threads are available in this pool, there is contention across requests and the overall system throughput falls. If the top level flows call subflows and activities concurrently, then this thread pool should be sized to accommodate the concurrency.

JDBC Connection pools

SAS Decision Services uses Apache DBCP connection pooling to connect to resources through JDBC. Currently, this includes databases and the SAS Federation Server. The former is used by the General I/O activity, while the latter is used by activities to execute DS2 code (includes scoring models). For SAS Federation Server, every connection on the Java client creates a service thread in SAS Federation Server that loads and compiles DS2 packages for execution. Loading and compiling DS2 packages are expensive in terms of the time taken to create the thread and the memory that it occupies.

Initializing the connection pool initializes the SAS Federation Server with threads that are already primed. Unlike the primary thread pool, the JDBC connection pool should be sized by the capacity and throughput of the database server or the SAS Federation Server it is referencing. For example, if there is no capacity on the database server, increasing the connection pool size merely moves the bottleneck to the database server. It does not improve the throughput.

Accessing the PoolDiagnostics.jsp Page

The PoolDiagnostics.jsp page can be displayed using any browser that can access the SAS Decision Services engine web application, and then by entering the URL `http://<IP address of engine server>:<port of engine web application server>/RTDM/PoolDiagnostics.jsp`

The page automatically refreshes every five minutes. If a more frequent refresh is required, use an appropriate value for the query parameter `refreshSeconds`. For example, using the URL `http://<IP address of engine server>:<port of engine web application server>/RTDM/PoolDiagnostics.jsp?refreshSeconds=2` causes the page to refresh every two seconds.

Items Displayed for Activity and Sub Flow Thread Pool

The following are displayed:

- A bar chart representing the primary thread pool of a specified engine.

- **Active threads** — Displays the number of active threads in the thread pool that are currently executing activities or sub flows. It is represented by a dark green bar in the bar chart. If the number of active threads are more than 90% of the maximum pool size, the color is set to red.
- **Pool size** — This is the size of the pool that includes both active and idle threads. Idle threads are shown as a medium green bar, not as dark as active and not as light as the largest pool size. If the size of the pool is more than 90% of the maximum pool size, the color is set to yellow.
- **Largest pool size** — This is the largest value of the pool size achieved since the engine process was started. This value is represented as a light green bar in the graph. This is the high water mark that indicates whether the present pool size is enough to deal with transient load peaks. If the size of the pool is more than 90% of the maximum pool size, the color is set to yellow.
- **Core pool size** — This is the configuration property that creates a set of threads in the pool when the engine is started. If this value is too low, the engine uses time initially to create the threads that are needed to support the initial load and might not be able to respond to initial spikes.
- **Max pool size** — This is the maximum configured size of the thread pool.

Items Displayed for JDBC Connection Pools

The following are displayed:

- **Resource name** — The name of the JDBC connection resource.
- **URL** — All of the server URL values for the named resource. SAS Decision Services supports multiple server URLs for a JDBC resource and distributes the load across them using a uniform random distribution. Every URL is associated with its own connection pool and for every URL the following are displayed:
 - A display bar representing the status of the connection pool. The relative sizes and colors are explained below.

- Created — A timestamp indicating when this pool was created. Pools might be invalidated and re-created if there are errors connecting to the server. The created timestamp can be used to diagnose issues around this.
- Active — The number of connections active in the connection pool. This is represented by a dark green bar. If this number is more than 90% percent of the maximum pool size, the color is set to red.
- High — The maximum number of connections that were in use since the pool was created. It is represented by a medium green bar. The value indicates any spikes in the load that have been encountered.
- Max active — The maximum number of connections that the pool is configured to hold.
- Idle — The number of connections that are idle in the pool. It is represented as a light green bar. Idle connections imply that the connection-specific objects have already been created in the database, or SAS Federation Server, and are ready for use.
- Max Idle — The configured maximum number of connections the pool can have before they are reaped and closed.
- Min Idle — The configured minimum number of connections the pool can have that the idle connection closing process will leave alone.

10

Custom Configuration

<i>Standard System Resources</i>	202
\$SAS_Activity_Resource	202
GeneralIO_Activity_Resource	202
\$User_Log_JDBCConnectionResource	202
<i>Configuring Additional Databases</i>	203
Overview	203
Database Installation Checklist	203
Install Database Client and Server Software	204
Define Database Users	205
Define a JDBC Connection System Resource	205
(Optional) Create Schema Aliases	205
<i>Configuring Additional SAS Federation Servers to Form a Server Cluster</i>	205
Overview	205
Install and Configure a New SAS Federation Server	206
Edit JDBC Connection System Resources	207
Clustering Best Practices	207
<i>Changing the Database Selection</i>	208
Overview	208
Oracle	208
SQL Server and DB2	209
General Steps	211

<i>Configure Access to SAS Data Sets</i>	212
<i>Moving DS2 Persistence to a Database Management System</i> ..	212
<i>Moving Application Data from a Web Infrastructure Data Server to Other Databases</i>	214

Standard System Resources

When your system was initially installed and configured, several system resources were created by default.

\$SAS_Activity_Resource

Configured to reference a SAS Federation Server for the purpose of executing SAS activities. SAS activity code is contained in DS2 packages. By default, these DS2 packages are stored in SAS data sets. It is possible to reconfigure the system to store activity DS2 packages in a third-party database, if you so choose. Contact your SAS on-site support personnel for assistance.

Note: Scoring models are published as SAS activities.

GeneralIO_Activity_Resource

Configured to reference a third-party database management system for the purpose of storing and accessing data. By default, this system resource is configured to reference the database that was chosen during configuration. Also, by default, the General I/O activity is configured to use this system resource.

\$User_Log_JDBCConnectionResource

Configured to reference a third-party database management system for the purpose of logging performance data. For more information, see [“Data Collection for Performance Analysis” on page 54](#). By default, this system resource is configured to reference the third-party database that was chosen during configuration.

Configuring Additional Databases

Overview

During installation, standard SAS Decision Services deployments are configured for access to one third-party database management system and for access to SAS data sets. (Optional) Access to additional third-party database management systems can be configured.

Note: These instructions assume that the additional database is to be used for data storage and access only, and not for use by SAS Federation Server to read DS2 packages.

Note: It is possible to access databases through a SAS Federation Server. However, doing so results in degraded performance. Instead, configure SAS Decision Services to use the native JDBC driver provided by your database vendor.

Your installation might include one or more development, test, or production environments. Repeat the procedures described in this section for each environment that you want to add the additional database to.

Database Installation Checklist

Complete the following checklist. This information is used in [“Install Database Client and Server Software” on page 204](#).

Table 10.1 SAS Decision Services Checklist — Database Information

Defaults

Database Type:

Choose One:

- ☐ DB2 9.5 or later
 - ☐ MS SQL 2008 or later
 - ☐ Netezza 7.0 or later
 - ☐ Oracle 10g or later
 - ☐ PostgreSQL 9.1.3 or later
 - ☐ Teradata 12.0 or later
 - ☐ SAS
-

Database Host:

Database Port:

Database Name:

Database User ID:

Database Password:

Install Database Client and Server Software

SAS Decision Services uses Apache DBCP for its JDBC Connection and statement pooling implementation. Therefore, the connection pooling services of the J2EE application server are not used.

The goal of installing the database client software is to make the native JDBC driver class for your database accessible to SAS Decision Services.

For SAS Web Application Server, you must copy the JDBC driver JAR files to the application server's lib directory.

To install the database client and server software:

- 1 Install the required database management system server software on a designated database server machine, and then configure your database server. See the installation documentation for the specific database.

- 2 Copy the JDBC driver JAR files to the `SAS-config-dir\Lev1\Web\WebAppServer\SASServer7_1\lib` directory.

Define Database Users

Defines a database user ID with create, alter, and delete table permissions that is used to access data from the database and is also used as the Java Access user ID.

Define a JDBC Connection System Resource

For more information, see [“Specify a New System Resource as a JDBC Connection” on page 72](#).

(Optional) Create Schema Aliases

For more information about creating schema aliases, see [“\(Optional\) Define a Schema Alias” on page 78](#).

Configuring Additional SAS Federation Servers to Form a Server Cluster

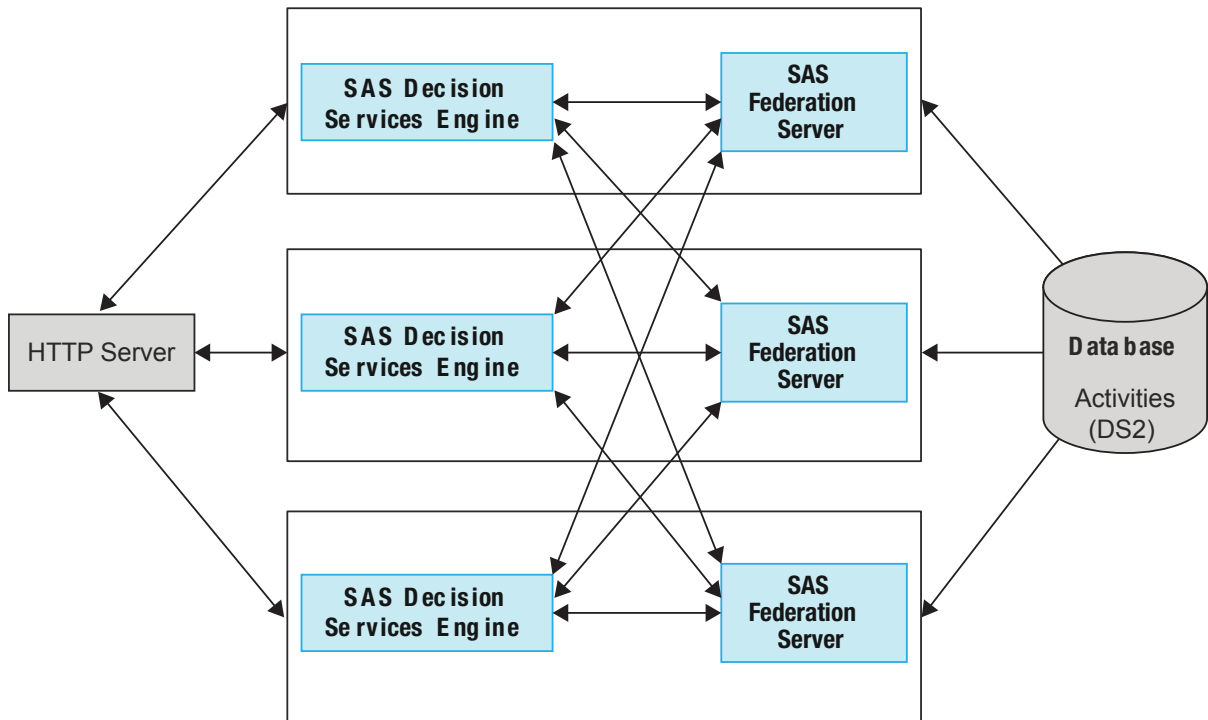
Overview

The standard SAS Decision Services installation and deployment process configures a single engine middle tier and a single SAS Federation Server. For production deployments, more than one middle-tier engine and collocated SAS Federation Server are typically configured to meet system performance and availability requirements. See [Best Practices on page 176](#) for information about how to determine the number of servers to allocate to each tier. A minimum of two middle tier SAS Federation Servers are required to support hardware failover.

The SAS Decision Services engine load balances every SAS Federation Server for which there is a corresponding URL in the JDBC Connection system resource that is

used for executing activities. Therefore, for proper operation, every such SAS Federation Server must have access to the same set of DS2 packages. This requires that every such SAS Federation Server be configured identically, with the same logins, database users, and DSNs. If your DS2 packages are stored in SAS data sets, those data sets must be located in a shared directory that is accessible from all such SAS Federation Servers.

Figure 10.1 Load Balancing



Install and Configure a New SAS Federation Server

See the *SAS Decision Services 6.4 Configuration Guide* for information about installing and configuring a SAS Federation Server. Be sure to configure the new SAS Federation Server identically to the one that is currently used to execute activities. It must have access to the same set of DS2 packages, and it must have the same logins, database users, and DSNs defined.

Edit JDBC Connection System Resources

Use the following procedure to edit your JDBC Connection System Resources to recognize the new SAS Federation Server that you configured earlier. If you are modifying a standard configuration, repeat this procedure for each of the system resources:

- `$SAS_Activity_Resource`
- `$Score_JDBCConnectionResource`
- `GeneralIO_Activity_Resource_FS` (only modify `GeneralIO_Activity_Resource` if you plan to use your new SAS Federation Server for SAS data set I/O).

Open SAS Management Console, select the **Folders** tab, and follow these steps:

- 1 Expand **System ► Applications ► SAS Decision Services ► Decision Services 6.4**.
- 2 Select the repository folder.
- 3 Right-click the desired system resource, and select **Edit System Resource** from the drop-down menu.
- 4 Modify the **Server URL** field only, by adding a space followed by the full URL, including protocol, of your new SAS Federation Server.
- 5 Click **OK** to save your changes.

Clustering Best Practices

- If DS2 packages are stored in SAS data sets (the default configuration), they must be located on a shared drive that is accessible by all SAS Federation Servers in the cluster. Otherwise, run-time errors will occur. Similarly, if the SAS Federation Server cluster is used to read data from SAS data sets, all servers must have access to the data sets on a shared drive.

- Multiple SAS Federation Servers, which are listed in the Server URL field of a system resource, are uniformly load balanced. Therefore, it is important to deploy each SAS Federation Server in a given cluster on the same class of hardware. If this practice is not complied with, the more powerful servers in the cluster will be under used and the less powerful servers might be overburdened.
- If DS2 packages are stored in a third-party database (a custom configuration), all SAS Federation Servers in the cluster must have access to the database and to the same DS2 packages.

Changing the Database Selection

Overview

To change the database selection for SAS Decision Services, you must install the required database management system client software on the SAS Federation Server and the SAS Server. For more information, see the installation documentation for the specific database.

Oracle

Add an entry into your TNSNAMES.ORA file and change the values that are shown in brackets to suit your environment. SAS uses addressname to connect to the database. SAS Federation Server and the JDBC connection system resources use sid to connect to the database. When defining this entry, define the addressname and the sid as the same value.

```
<addressname>=
(DESCRIPTION=
  (ADDRESS_LIST=
    (ADDRESS= (PROTOCOL=TCP) (Host=<hostname>)
    (Port=<port>)))
(CONNECT_DATA=
  (SERVICE_NAME=<sid>)
))
```

SQL Server and DB2

Overview

If you want to run the automated configuration of the user log tables, create the ODBC data source names on the SAS Federation Server and the SAS Server, before you run the SAS Deployment Wizard configuration. After that, create the ODBC data source name as an administrative user, and use the native database driver. The steps for creating these data source names vary depending on the operating system.

Windows

- 1 From the **Start** menu, navigate to **Control Panel ► Administrative Tools ► Data Sources (ODBC)**.
- 2 On the **System DSN** tab, click **Add**.
- 3 Select the driver that corresponds to your database, and click **Finish**.
- 4 Complete the options below based on database type.

SQL Server

Data Source Name

Enter the data source name.

Description

This is optional.

Server

Enter host for the SQL server database.

With SQLServer Authentication

Enter user ID and password.

You can change default database

This is optional

You can change the log location

This is optional.

Select Test Data Source

Select the data source.

DB2

Data Source Name

Enter the data source name.

Description

This is optional.

Database Alias

Select **ADD**.

Data Source tab

Enter the user ID and password.

TCP/IP tab

Enter the information for each field.

5 Test the connection on each DSN, and click **Finish**.

UNIX or Linux

Use the interactive ODBC Configuration Tool, `dfdbconf`, to add new data sources to the ODBC configuration.

- 1** From the root directory of the SAS Federation Server installation, run: `./bin/dfdbconf`
- 2** Select **A** to add a data source. You can also use `dfdbconf` to delete a data source.
- 3** Select a template for the new data source by choosing a number from the list of available drivers.
- 4** You are prompted to set the appropriate parameters for that driver. The new data source is then added to your `odbc.ini` file.

Once you have added all of your data sources, the interactive ODBC Viewer application, `dfdbview`, can be used to test your connection, as shown in the following example:

```
./bin/dfdbview my_odbcdsn
```

For non-ODBC connections, use the vendor-supplied client configuration utility. You might be prompted for a user name and password. If the connection succeeds, you will see a prompt from which you can enter SQL commands and query the database. If the connection fails, SAS Federation Server displays error messages describing the reasons for the failure.

General Steps

To complete the database selection change after you have completed the database-specific steps:

- 1 Re-create the batch, monitoring, and user log tables in the new database. The scripts for each table are located in the install directory of the SAS Decision Services server configuration. The path is **Program Files\SASHome\SASDecisionServicesServerConfiguration\6.4\Configurable**.
- 2 Copy the new JDBC JAR files into the application server lib directory.
- 3 Using SAS Data Management Studio, create a new database domain, and database user, if the user and domain are different for the SQL Server database. Add a login to the SAS Federation Server administrative user for this new database user, if applicable.
- 4 From the SAS Federation Server Manager, log on as the administrator and create a new database DSN that points to the new database. For more information see *SAS Federation Server Administrator's Guide*.
- 5 From the Decision Services Manager plug-in for SAS Management Console, edit the \$UserLog and General IO Resource to point to the new database DSN.

Configure Access to SAS Data Sets

To access SAS data sets from SAS Decision Services, create a system resource that references the SAS Federation Server that you intend to use for data set I/O, and then create a General I/O instance that references it.

Note: If you are using SAS Real-Time Decision Manager, you can accomplish this by creating a data process through the Customer Intelligence plug-in for SAS Management Console.

Except under special circumstances, SAS data sets should be used only for reading data. It is possible to create decision services that write SAS data sets, but in general this practice should be avoided. SAS Decision Services is multi-threaded, and capable of executing multiple Read or Write operations concurrently. SAS data sets have file-level locking, so attempts to write data sets from multiple concurrent threads results in deadlocks and possibly loss of data. If you must write to SAS data sets, then set the connection pool values of both MaxActive and MaxIdle to 1 in the appropriate system resource. This causes I/O operations to be serialized but to perform slowly. If you must write data, the use of one of the supported database management systems is highly recommended.

Moving DS2 Persistence to a Database Management System

The following steps enable you to publish DS2 packages to a database management system, rather than to SAS data sets.

- 1 From Data Management Studio, add the database domain and then log in as the system user or as the user who is defined in the \$SAS_Activity_Resource.
 - a Double-click the **Administration** menu bar.
 - b Click plus (+) to expand the **SAS Authentication Server** menu.

- c** Double-click your SAS Authentication Server definition.
- d** Log in with the system user ID.
Note: On Windows, use the fully qualified domain and user ID.
- e** Once you have logged in, right-click **All Domains**, and select **New Domain**.
- f** Enter a domain name for the database user.
- g** Under Domains, select **All Domains ► database_domain**.
- h** (Optional) Enter a description.
- i** Enter a user name format for the database user. Click the appropriate radio button based on the database server platform.
 - Windows users select the radio button down-level log-in name.
 - UNIX users select the radio button user name only.
- j** Click **OK**.
- k** Create the database user's log-in information.

- 2** Use SAS Federation Server Manager to create a database DSN and DataService, if you do not have one. Make sure you have installed the database management system client on SAS Federation Server. You can also modify the following SAS script to create the DSNs and DataService. Each database management system SAS program is located in the installation directory found in **Program Files \SASHome\SASDecisionServicesServerConfiguration \6.4\Configurable\Utilities**.

```
PROC FEDSQL server=&server protocol=bridge port=&fedport uid="&authuid"
  pwd="&authpw" conn="(DRIVER=SYSCAT) ";

CREATE DATA SERVICE &dataservice TYPE ORACLE CATALOG &catalog DOMAIN
  &domain {OPTIONS ( CONOPTS ( DRIVER ORACLE, PATH &path) ) };

CREATE DSN &feddsn under &dataservice NOPROMPT 'DRIVER=ORACLE' AS
  ADMINISTRATOR;
```

QUIT;

- 3 Add the new database DSN as the default DSN, by making it the first DSN on the list. To do this, remove BASE_DSN, and add the new database DSN. Then, add the BASE_DSN back again.
- 4 Modify the connection lib of the tap packages that are in the loadutilpackages.sas program to point to the new database DSN.

```
proc ds2 nolibs noprompt="driver=remts;server=<hostname>;port=21032;
  protocol=bridge;uid=<systemuser>;pwd=<password>;
  conopts=(driver=ds2;conopts=(driver=sql;conopts=(DSN=DATABASE_DSN)))";
```

Moving Application Data from a Web Infrastructure Data Server to Other Databases

The SAS Decision Services database includes tables to manage user logs, and batch and monitor application data. However, if you decide to change databases, you must manually migrate the information. To do this:

- 1 Stop the application server where the SAS Decision Services monitor is located (such as SASServer7_1).
- 2 Locate the file *<SAS-configuration-directory>/Lev1/Web/WebAppServer/SASServer7_1/conf/server.xml*.
- 3 In the server.xml file, locate this resource: `<Resource auth="Container" driverClassName="org.postgresql.Driver" factory="com.atomikos.tomcat.NonXABeanFactory" maxPoolSize="100" minPoolSize="10" name="sas/jdbc/DecisionServices" password="$ {pw.sas.jdbc.DecisionServices}" testQuery="select 1" type="com.atomikos.jdbc.nonxa.AtomikosNonXADataSourceBean" uniqueResourceName="sas/jdbc/DecisionServices" url="jdbc:postgresql://hostname.domain.com:9432/DecisionServices" user="DecisionServices"/>`

- a** Modify the host name to point to the host and port of the database server.
 - b** Modify the `driverClassName` from `org.postgresql.Driver` to the JDBC driver class name for the database.
 - c** Change the URL to the appropriate URL for the database vendor.
- 4** On the SAS Decision Services engine server, locate the corresponding SQL file `<SASHome>\SASDecisionServicesEngineServer6.4\Config\Deployment\Data\<database>` and copy it to a location on the target database server. Edit this file to modify the password for the DecisionServices user.
 - 5** Using the appropriate database vendor SQL utilities, execute the `create-dcsv-tables-manual.sql` file to load tables.
 - 6** Edit the `data-topology.sql` file to include the web services URL for the SAS Decision Services engine, and then execute this SQL file.
 - 7** Encrypt the database password using a similar command: `C:\Program Files\<SASHome>\SASWebApplicationServer\9.4>java -cp tomcat-7.0.30.A.RELEASE\lib\tcServer.jar;tomcat-7.0.30.A.RELEASE\bin\tomcat-juli.jar;tomcat-7.0.30.A.RELEASE\lib\tomcat-coyote.jar com.springsource.tcserver.security.PropertyDecoder -encode This! sTheSpringSourcePassphrase <password>`
 - 8** Modify the `catalina.properties` file located in `<SAS-configuration-directory>\Lev1\WebAppServer\SASServer7_1\conf` and add this encrypted password to this property `pw.sas.jdbc.DecisionServices=.`
 - 9** Modify the `$User_Log_JDBCConnectionResource` system resource with the following information:
 - Driver Class: Enter the database JDBC driver class.
 - Server URL: Enter the database JDBC URL.
 - Enter the user name and password for the database.

- 10** Copy the database vendor JDBC JAR files into the lib folder `<SAS-configuration-directory>\ Lev1\WebAppServer\SASServer7_1`
- 11** Start the application server where the SAS Decision Services monitor is located. (such as SASServer7_1).

Appendix 1

Database Requirements

The following table shows the SAS Federation Server requirements:

Table A1.1 Supported Database Clients

Database Client	Native Driver	Specific Version
DB2 V8.2 Fixpack 9 client and later	DB2	DB2 10.5 client
Oracle 10g client and later	Oracle Wire Protocol	Oracle 11.2.0.2 client and administrator and Oracle Enterprise Manager
TTU 12 and later (client and utilities, including TPT)	Teradata	Teradata Client 14.0
Microsoft SQL Server 2008 and later	SQL Server Classic Wire Protocol	SQL Server Native Client 10.0, Greenplum 6 (Data Direct Branded Driver 4.2.2)

Here are the SAS 9.4 requirements:

- DB2 Universal Database, Version 8.1 FixPak 18 or later (64-bit libraries)
- Microsoft SQL Server requires a 64-bit ODBC driver
- The minimum required Oracle Client release is Oracle, Release 10g (64-bit libraries)
- Teradata Database 12 or later, Teradata CLiv2 client libraries, TTU 12 or later

Appendix 2

Logs and Troubleshooting

Troubleshooting

Troubleshooting problems with SAS Decision Services requires inspecting the logs generated by the following components:

- SAS Decision Services design server
- SAS Decision Services engine server
- J2EE Application Server
- SAS Management Console
- SAS Metadata Server
- Stored Process Server
- SAS Federation Server
- SAS Authentication Server

Log File Locations

The following table summarizes the location of log files generated by the various SAS Decision Services components.

Table A2.1 Log Locations

Component	Default Location of Logs
Compute Tier	<p><SAS Config> is the location for installing the configuration for SAS. Here is an example: D:/SAS/Config/Lev1</p> <p>yyyy-mm-dd is the date or time of creation of the log file.</p> <p>nnnn is a four-digit number to differentiate between logs created by multiple processes that perform the same function.</p> <p>N.N is the major and minor version number of SAS Decision Services.</p>
Metadata Server	<p><SAS Config>/SASMeta/MetadataServer/Logs/SAS_Meta_MetadataServer_YYYY-MM-DD_NNNN.log</p>
SAS Federation Server	<p><install root>\FederationServer\4.1\etc\dfs_log.xml</p> <p><install root>\FederationServer\4.1\var\log</p>
Middle Tier	
SAS Decision Services Engine Server	<p><SAS Config>/Web/Logs/SASServer7_1/SASDecisionServicesEngineServerN.N.log</p>
SAS Decision Services Design Server	<p><SAS Config>/Web/Logs/SASServer7_1/SASDecisionServicesDesignServerN.N.log</p>
SAS Decision Services Monitor	<p><SAS Config>/Web/Logs/SASServer7_1/SASDecisionServicesMonitorN.N.log</p>
SAS Web Application Server	<p><SAS Config>\Web\WebAppServer\SASServer7_1\logs\server.log</p>

JDBC Database I/O

To troubleshoot JDBC database I/O, your SAS Technical Support representative can assist you with configuring logging settings that capture information about the I/O that is being performed. These settings affect performance. Therefore, they should be enabled only during troubleshooting. When logging is set to DEBUG level, the log message contains the prepare time, the execution time, and the SQL that is used. When logging is set to TRACE level, the log message includes everything from the DEBUG message plus the values used in the JDBC call. Depending on the call, these can include input values, result values, and WHERE clause values.

Appendix 3

Batch Execution

Overview

Batch execution provides the following capabilities:

- The batch execution of transactions stored in database tables
- High speed simulations
- Application and system performance data

Batch execution is needed in the design environment as well as in test and productions. Batch execution logic resides in the SAS Decision Services engine, in the form of a simple batch driver. Access is obtained through a web service interface.

Locating batch processing inside SAS Decision Services has the following advantages:

- It allows message formatting and parsing overhead to be eliminated, in both directions, by reading transactions directly into the internal event objects. Similarly, results are written directly to the output tables without formatting and sending XML messages.
- It allows SAS Decision Services to collect monitoring data (node-level hit counts and system performance metrics) as part of the batch run, and to write this data to a convenient output table.

To change the execution mode for the engine and monitor, from the SAS Management Console **Plug-ins** tab navigate to **Application Management ► Configuration Manager ► SAS Application Infrastructure**. Then, right-click **SAS Decision Services**

Engine Servers 6.4 and select **Properties**. On the **Advanced** tab, change the `sasds.engine.execution.mode` property value to either `REALTIME` or `BATCH`.

Note: If the execution mode has been changed, the system must be restarted for the change to take effect. The system includes the monitor and the engine. In the case of a cluster, all the engines must be shut down completely and restarted.

CAUTION! SAS Decision Services interprets DateTime values that are stored in a database or in SAS data sets, as being in the Greenwich Mean Time (GMT) time zone. A default time zone (GMT) is associated with values that are read from a database because time zone information is not persisted with the data. As a best practice, always store DateTime values using GMT time zone. This practice enables compatibility with SAS Decision Services and removes ambiguity.

Design-Time Simulations

Overview

Two methods have been added to the design server to support simulations in the design environment. The design server interface is accessed through the Design Server Factory by passing in the session ID and the repository name.

`submitSimulation()`

```
/**
 *
 * @param flows
 * @param simulationDescription
 * @return
 * @throws FaultMessagesHolder
 */
long submitSimulation(List<FlowDefinition> flows,
    SimulationDescriptionType simulationDescription) throws DesignServerException;
```

The method `submitSimulation` accepts a set of flows as JAXB objects and a simulation description that holds parameters for the simulation, as described below.

If the simulation call is accepted, a positive simulation ID is returned and the simulation is started. The call to submit simulation is non-blocking, which means that it does not wait for the simulation to finish. The reason that it is non-blocking is that the simulation call is designed to execute a large number of transactions that can easily time out the

HTTP call. Only a single simulation can be executed per session. If a submitSimulation call is made while one is already in progress, a value of -1 is returned.

Like the test method, the list of flows is scanned for dependencies (such as events, global variables, and activities). These dependencies are loaded into an isolated testing environment from the repository for executing the simulation.

The simulation ID that is returned is unique while the Design Server process is running. After the Design Server is restarted, the simulation ID is reset and starts counting from 1 again. It is possible to start the count at a higher number by changing the configuration of the design server. The transaction input data is read from the inTable. The transaction output data is written to the results table, and the hit counter values for each node and events are written out to the stats table.

Parameter	In/ Out	Type	Description
eventName	In	String	The event for which transactions are provided in the input table.
inTable	In	String	The name of table containing input transactions. The columns of inTable must match the names and types of the request variables of the event named by eventName. There is also the correlation_id column, which is used to match response records with transaction records.
outLibrary	In	String	The library or schema containing the output transactions table. This refers to a SAS Decision Services library resource entry.

Parameter	In/ Out	Type	Description
outTable	In	String	<p>The name of the table to insert transaction results into (must be created in advance). The columns of outTable must match the names and types of the reply variables of the event. There are also three additional columns:</p> <ul style="list-style-type: none">■ Column 1 must be of long type and named batch_job_id.■ Column 2 must be of string type and named correlation_id, with a length of 32.■ Column 3 must be of character type, with a length of 8, and be named status_cd. <p>Column batch_job_id is populated with the simulationId value passed in to identify the set of records belonging to the simulation run. Column status_cd indicates the success or failure of the transaction. Successful transactions have the value "OK" in status_cd. Unsuccessful transactions have the value "ERROR."</p> <p>The remaining columns must match the event reply variable names and types. Missing request or response columns are tolerated, but at least one of each must be present or an error is returned.</p>
statsLibrary	In	String	<p>The library or schema containing the node-level counts table. This refers to a SAS Decision Services library resource entry.</p>

Parameter	In/ Out	Type	Description
statsTable	In	String	Name of the table to insert hit counts into (must be created in advance). The precise schema must appear in the Table Definitions section of the statistics table. For more information, see “Table Definitions” on page 239 .
recordingOptions	In	BatchRecording	There are two options: APPEND or OVERWRITE. This parameter specifies whether the output results and statistics are appended to the results table and stats table, respectively, or whether the results and stats tables are cleared before the simulation is run. Because the simulation ID is also written out as a column value of the table, it is possible to hold output data from multiple simulations in the same table.
threadingOptions	In	BatchThreadingType	There are two options: SINGLE_THREADED or MULTI_THREADED. The SINGLE_THREADED flag restricts batch processing to a single thread. If a SAS data set is specified for outTable, SINGLE_THREADED must be set to prevent I/O contention, due to SAS file-level locking. The default value is MULTI_THREADED. Having SINGLE_THREADED enabled might affect performance.
timeZoneOptions	In	BatchTimeZoneOverride	There are three options: NONE, GLOBAL, or COLUMN.

Parameter	In/ Out	Type	Description
timeZoneColumnOrId	In	String	This parameter is ignored (can be null) when <code>timeZoneOptions = NONE</code> . When <code>timeZoneOptions = GLOBAL</code> , it specifies the time zone identifier to use as the client timezone for the entire simulation. When <code>timeZoneOptions = COLUMN</code> , it specifies the column of <code>inTable</code> containing the timezone identifier to use as the client time zone for the corresponding transaction.

getSimulationStatus

```
/**
 *
 * @param simulationId
 * @return
 * @throws FaultMessagesHolder
 */
SimulationStatusType getSimulationStatus(long simulationId)
    throws DesignServerException;
```

The status of a simulation can be queried using this method. The method accepts a simulation ID and returns a JAXB object containing the details of the status of the simulation.

If the simulation ID is not recognized by the design server, a null value is returned. This could happen if the simulation with that ID has not been accepted yet, it was accepted in a different session, or the simulation status is no longer held in memory. The status of at most one simulation is maintained per session. A new simulation run replaces the status in the given session.

The status values are also stored in the stats table. These can be retained by using the APPEND recording option, for subsequent simulation requests. They can be retrieved by selecting records with `batch_job_id` matching the `simulationId` of a given simulation. When using APPEND, the client application is responsible for deleting records that are no longer needed from the output tables.

The major components of the status object are described in this table:

Parameter	Type	Description
state	INITIALIZING, IN_PROGRESS, COMPLETE, and FATAL_ERROR	This value reflects the state of the simulation. After a simulation is submitted, it goes through an initialization phase when the tables are checked for correctness. The state changes to IN_PROGESS when transactions are processed. After all transactions are processed, the state is set to COMPLETE. If the simulation is not started because of a serious error (for example, database tables could not be accessed) the state is set to FATAL_ERROR. If a simulation is submitted in the session, the state value is always available.

Parameter	Type	Description
transactions counts	Three long values	<p>The transaction counts are three long numbers that represent the total number of transactions, the number of completed transactions, and the number of error transactions. These are not meaningful if the state is <code>INITIALIZING</code>. During the <code>IN_PROGRESS</code> phase, these numbers reflect the actual transactions processed and can be used to track progress. If the state is <code>COMPLETE</code>, the total number of transactions should be a sum of the number completed and the number that had errors. Generally, the <code>FATAL_ERROR</code> can be entered during the initialization phase. In this case, the counts are all zeros. If a fatal error is encountered during processing, the state is set to <code>FATAL_ERROR</code> and the counts reflect the transactions that were processed when the fatal error took place.</p>

Parameter	Type	Description
hit counts	Three map objects	These include three map objects. The first two contain hit counts by events and nodes. These are map objects that have a string type as the key and a long type as the value. The Event Map maps the actual event name to a number that indicates the number of times the event was invoked during the simulation. The Node Map maps a compound node name scoped by the flow name (in the format: flow name.node name) to a number that indicates the number of times the node was invoked during the simulation. The third map maps the compound node name for activity call nodes to the activity and the activity method that they invoke. The hit counts are returned only after the simulation is in the COMPLETE state.

Run-Time Batch Interface

Overview

Run-time batch processing follows the identical rules and assumptions used in a real-time production. All referenced top-level decision flows must be active. All required artifacts referenced by active flows and sub-flows must be available.

batchRun()

The batchRun method executes the transactions contained in inTable, writes transaction responses to outTable, and saves statistics (counts and execution times) in statsTable.

The columns of `inTable` must match the names and types of the request variables of the event named by `eventName`. There is an additional column that must be of type string, length 32, and named `correlation_id`. The `correlation_id` column is used to match requests with responses.

The columns of `outTable` must match the names and types of the reply variables of this event. There are three columns for `outTable`.

- Column 1 must be named `jobId` and be of long type.
- Column 2 must be of string type, length 32, and named `correlation_id`.
- Column 3 must be of character type, length 8, and named `status`.

The remaining columns must match the event reply variable names and types. The `jobId` column is populated with the unique job identifier passed to `batchRun()`. The `correlation_id` column is populated with an identifier that matches the corresponding input transaction. The `status` column indicates success or failure of the corresponding transaction. Successful transactions have the value `OK` in the `status` column. Unsuccessful transactions have the value `ERROR`.

```
batchRun(jobId, eventName, inLibrary, inTable, outLibrary, outTable,
statsLibrary, statsTable, recordingOptions, threadingOptions)
```

Parameter	In/Out	Type	Description
jobId	In	Long	A unique job ID to associate with this batchRun. (This passes jobId to subsequent status queries. It is used as the key field of output records).
eventName	In	String	The event for which transactions are provided in the input table.

Parameter	In/Out	Type	Description
inLibrary	In	String	The Library or scheme containing the input transactions table. This refers to a SAS Decision Services library resource entry.
inTable	In	String	The name of the table containing input transactions.
outLibrary	In	String	The library or schema containing the output table. This refers to a SAS Decision Services library resource entry.
outTable	In	String	The name of table to insert transaction results into (this must be created in advance).
simulationDateOptions	In	BatchSimulationDateOverride	There are three options: NONE, GLOBAL, or COLUMN.

Parameter	In/Out	Type	Description
simulationDateColumnOrLiteral	In	String	This is ignored (can be null) when simulationDateOptions = NONE. When simulationDateOptions = GLOBAL, it specifies the datetime value to use as the simulation time for the entire batch run. When simulationDateOptions = COLUMN, it specifies the column of inTable that contains the datetime to use as the simulation time for the corresponding transaction.
simulationDateTimeZoneColumnOrId	In	String	This is ignored (can be null) when simulationDateOptions = NONE. When simulationDateOptions = GLOBAL, it specifies the time zone identifier to use as the simulated client timezone for the entire batch run. When simulationDateOptions = COLUMN, it specifies the column of inTable that contains the timezone identifier to use as the simulated client time zone for the corresponding transaction.

Parameter	In/Out	Type	Description
statsLibrary	In	String	The library or schema containing the summary statistics or node-level counts table. This refers to a SAS Decision Services library resource entry.
statsTable	In	String	The name of the table to insert summary statistics or counts into (this must be created in advance).
recordingOptions	In	BatchRecordingType	There are two options: APPEND or OVERWRITE.
threadingOptions	In		There are two options: SINGLE_THREADED or MULTI_THREADED.
timeZoneOptions	In	BatchTimeZoneOverride	There are three options: NONE, GLOBAL, or COLUMN.

Parameter	In/Out	Type	Description
timeZoneColumnOrId	In	String	This is ignored (can be null) when <code>timeZoneOptions = NONE</code> . When <code>timeZoneOptions = GLOBAL</code> , it specifies the time zone identifier to use as the client timezone for the entire batch run. When <code>timeZoneOptions = COLUMN</code> , it specifies the column of <code>inTable</code> that contains the timezone identifier to use as the client time zone for the corresponding transaction.

OVERWRITE causes the results and statistics tables to be cleared before job execution. The default is APPEND, which causes new output to be appended to any existing output.

SINGLE_THREADED restricts batch processing to a single thread. If a SAS data set is specified for `outTable`, SINGLE_THREADED must be set to prevent I/O contention because of SAS file-level locking. The default value is MULTI_THREADED. Having SINGLE_THREADED enabled might affect performance.

getBatchStatus()

The method `getBatchStatus` returns the total number of input transactions, the number completed so far, and the number of errors encountered so far, in the current or most recent batch job identified by `jobId`. It also returns the hit counts and performance data.

```
getBatchStatus(jobId)
```

Parameter	In/Out	Type	Description
jobId	In	Long	Unique batch job identifier.
status	Out	BatchStatusType	An object containing the current job state (IN_PROGRESS, COMPLETE, or NOT_LOADED); start time, elapsed time (if applicable), end time (if applicable), hit counts, and performance data.

Activation Rules

In run-time environments (test and production), at most one flow per event can be active at any given time.

In practical terms, this rule allows multiple batch jobs to be executed concurrently using a single SAS Decision Services engine. It also supports realistic batch simulations of real-time multi-flow applications. As with real-time execution, batch jobs are not isolated from one another. For example, two flows can write to the same table or can call a common sub-flow concurrently.

Execution Modes

In general, mixing batch and real-time execution does not work well, for the following reasons:

- It can render simulation results non-deterministic and therefore inconclusive.
- Batch processes starve real-time processing of resources, causing service level agreement violations. (This is the same reason that dedicated hardware is strongly recommended in a production environment.)

- It makes hardware capacity planning nearly impossible, yielding either unreasonably wide deviations in service level agreement guarantees or excessive hardware capacity requirements.

Except in rare circumstances, a given environment should execute in batch processing mode or in realtime transaction processing mode, but not in both at the same time.

Processing Mode	Description
Realtime	Active flows are ready to process events from inbound channels (either directly or indirectly, as sub-flows), and do not accept batch processing requests.
Batch	Active flows are ready to participate in batch simulations (either directly or indirectly, as a sub-flow), and do not listen to inbound channels.

Processing mode is set on a given development, test, or production environment through the SAS Decision Services SAS Management Console plug-in. When an environment is in real-time mode, it processes events arriving at the web service endpoint. However, it rejects any batch processing requests. When an environment is in batch mode, it processes batch requests, but ignores inbound events.

The default processing mode is batch for the design-time batch engine and realtime for all other environments. This setting has no affect on the design server.

Record Formats and Restrictions

Overview

The input transactions table must contain one transaction per record. After batch execution is complete, the results table contains one record per transaction response, with the columns that match a response variable (name and type) populated with response data. A correlation_id column is included in both tables for matching transactions records with results records.

Missing transaction and response columns are tolerated, as are extra columns. However, for batch processing to succeed, at least one column of the transactions table must match an event request variable. Also, at least one column of the results table must match an event response variable.

If the number of input or output columns exceeds the database column limit (1000 columns on Oracle, for example), then SAS data sets must be used. If a SAS data set is used for outTable, you must specify the option SINGLE_THREADED on the call to batchTest or batchRun to prevent I/O deadlock.

Table Definitions

The following DDLs lists the required columns of each table. Compatible column types and narrower column widths can be used on a case-by-case basis. However, attempts to insert data that is longer than the destination column width result in run-time errors. The names of required columns must match the names given below.

Example Code A3.1 Transaction Table (inTable) DDL

```
CREATE TABLE DS_BATCH_TEST (
  correlation_id      VARCHAR(32),
  <first request variable name & type>,
  <second request variable name & type>,
  <etc.> )
```

Example Code A3.2 Results Table (outTable) DDL

```
CREATE TABLE DS_BATCH_RESULTS (
  batch_job_id      VARCHAR(32),
  correlation_id     VARCHAR(32),
  status_cd         VARCHAR(10),
  <first response variable name & type>,
  <second response variable name & type>,
  <etc.> )
```

Example Code A3.3 Statistics Table (statsTable) DDL

```
CREATE TABLE DS_BATCH_STATS (
  batch_job_id      VARCHAR(32),
  flow_node_nm      VARCHAR(250),
  activity_nm       VARCHAR(250),
  method_nm        VARCHAR(250),
  entity_type_nm    VARCHAR(250),
  hits_cnt          NUMBER(12),
  average_latency_ms_value DECIMAL(18,5),
  timestamp_dttm    TIMESTAMP)
```

Disabling Writing Batch Results

Simulations, or batch jobs, are sometimes executed for the sole purpose of generating hit counters. To accommodate usage, SAS Decision Services allows batch job results to be suppressed. Suppression is optional. The mechanism for suppressing results is similar to the mechanism for suppressing statistics output.

To prevent results from being written to disk when running a design-time simulation, client applications can set one or both of the following `submitSimulation()` arguments to NULL:

- `outLibrary`
- `outTable`

To prevent results from being written to disk when running a production batch job, client applications can set one or both of the following `batchRun()` arguments to NULL:

- `outLibrary`
- `outTable`

When results are suppressed, the other functionality is unaffected, and transactions are processed normally.

Appendix 4

Activate Flows Using BatchActivator

Note: BatchActivator has been deprecated for SAS Decision Services 6.4. It is still supported for activating or deactivating decision flows. However, it is preferred that you use the SAS Decision Services Administration API through a command-line REST utility. For more information, see [Appendix 6, “SAS Decision Services Administration API,” on page 275](#).

BatchActivator is a command-line utility that is used to activate or deactivate decision flows. It can be used either stand-alone or in scripts. The utility requires connecting to the SAS Metadata Repository as part of its operation. The connection information is provided to the utility through command-line parameters. Also, user credentials used by the utility are supplied as command-line parameters, including supplying the credentials in a separate profile file.

When it is scripting, the utility returns a completion code of 0 to indicate success and 8 to indicate an error.

Multiple flows can be activated or deactivated at a time, with the following parameters:

- The input file must contain the list of flow names to be activated or deactivated, one name per line.
- All flows that are specified in the input file must be in the same state. The state can be either inactive (for activation) or active (for deactivation). Otherwise, you receive an error.

The utility updates changes in the repository and notifies the engine about the changes that it makes. If the engine is not reachable by the utility, the flows are deactivated in the repository. However, the utility also supports an option called OFFLINEOK. If the engine

is not reachable and you specify the OFFLINEOK option, the changes in the repository are not rolled back.

The following information is logged by the utility:

- The names of all flows that are activated or deactivated.
- Any validation errors that prevent activation .

The following usage statement is printed when the utility is run with the -help option or when the command-line parameters are incorrect:

Note: The English version of the switch must be used, even if the description has been translated.

```
BatchActivator [-?] [-a] [-d] [-debug] [-domain <domain>] [-f <file name>]
[-host <hostname>] [-l] [-log <log-file>] [-nolog] [-o] [-password <password>]
[-port <port>] [-profile <profile>] [-user <userid>]
```

Options	Descriptions
-?, -help	Prints help information.
-activate	Activates the flows.
-debug	Prints debugging information.
-domain <domain>	Provides user authentication domain information.
-f <file name>	The file that contains the names of the flows to activate or deactivate.
-host <hostname>	Metadata server host. Required if -profile is not set.
-log <log-file>	Log file or directory.
-nolog	Disable log file.
-o, --offline	Continue if the engine is off line.

<code>-password <password></code>	User login password. Required if -profile is not set or if the profile does not contain connection credentials.
<code>-port <port></code>	Metadata server port. Required if -profile is not set.
<code>-profile <profile></code>	Metadata server connection profile. Can be used in place of the -host, -port, -user, and -password options.
<code>-user <userid></code>	User login identity. Required if -profile is not set or if the profile does not contain connection credentials.

Appendix 5

REST API for SAS Decision Services Transaction Processing

<i>Overview</i>	246
<i>Latency Requirements</i>	247
<i>Use Cases</i>	247
<i>Other Important Features</i>	248
<i>Terminology</i>	248
Decision Services	248
Decision Services Artifacts	249
Decision Services Engine	249
Decision Services Environment	249
Validity Rules	249
Administrative Services	250
Activation	250
Run-time Services	250
Decision	250
Decision Request	250
Decision Flow	251
Decision Variable	251
Decision Definition	251
Bindings (input and output)	251
Decision ID	252
Time Out	252

Job	252
Pagination	252
Overview	252
Guidance	253
Media Types	254
Externally Defined Media Types	254
SAS Decision Services Media Types	255
Resources	268
Collection /	268
Collection /decisionDefinitions	269
Collection /jobs	271
Resource /descisionDefinitions/{decisionID}	272
Resource /decisions/{decisionId}	273
Post-Installation Steps	274

Overview

This API allows access to decisions that are generated by the SAS Decision Services engine. Specifically, the API accepts inputs that are represented as decision parameters, processes them in the engine, and returns a representation of the decision generated. The API is implemented as a Representational State Transfer (REST) interface that accepts JSON payloads.

The REST API generates decisions in the SAS Decision Services engine. To generate a decision, a client application posts a decision request object to a named decision definition and receives a decision in response. The request includes a set of bindings (one or more name-value pairs) that serve as inputs to the decision generation process. Each binding has a name that is a string and a value. The value might be a number, a Boolean value, a string, an array, or a table. The generated decision object contains a set of bindings that are used by the client applications to implement the decision. For example, a call center might use a decision that contains items to offer to a customer for sale. Each decision definition has a required set of bindings with fixed names and types

for the decision request as well as a defined set of bindings for the corresponding decision object that was produced.

A decision services engine might hold several decision flows. Each such flow contains code that generates a decision object when it is executed. Each flow is associated with only one decision definition that determines the inputs and outputs and that is an interface to the flow. The client application does not reference the flow directly. Rather, it uses the decision name that is defined in the decision definition. When a decision request is received by the engine, it looks up the decision definition with the supplied decision name, finds the flow that is associated with the decision definition, executes the flow, and returns the generated decision.

Latency Requirements

In most cases, the decision must be returned to meet strict latency constraints (order of 10–100s of milliseconds) that require a synchronous call to the engine. In other cases, the client application does not care about the decision generated, relying instead on the side-effects of executing a decision flow. In this case, the client application can execute the decision flow asynchronously. The post returns immediately and does not block the client application. The side effects are completely determined by the decision flow.

Use Cases

A typical use case is a call center application that sends information to the engine to determine the marketing offer to return to the caller. The call center application typically sends the caller's identification and some information that is not already available in the application's data storage (for example, the reason for the call). The engine might retrieve additional caller-specific information from the database and execute decision flows to generate the offer.

Another use case might involve a website application that sends information to the engine in order to retrieve offer information that drives banner advertisements on the website. The decision is generated when creating content for the website and any delay

in generating the page can cause the customer to leave the site. Therefore, a timely response within strict latency requirements is critical.

Other Important Features

Here are additional important features of the REST API:

- 1** The decision is modeled as a transient resource that is created for every decision request. Therefore, there are no available GET methods to access decisions that have already been generated.
- 2** The representations for decisions and decision requests are implemented as separate media types.
- 3** The current implementation supports only JSON payloads.
- 4** The POST method on the decisions collection returns a representation of generated decision synchronously.
- 5** The POST method on the jobs collection returns only an accepted status and internally queues the decision flow for execution using the supplied decision request.

Terminology

The following terms are applicable to the REST API for SAS Decision Services Transaction Processing as well as the [Appendix 6, “SAS Decision Services Administration API,” on page 275](#).

Decision Services

A collection of services that allow external and internal applications to generate decisions.

Decision Services Artifacts

The operation of an engine is controlled by the set of artifacts that are loaded in the engine. These artifacts include decision definitions, decision flows, and variables. The artifacts are created using design-time services in a decision services environment.

Decision Services Engine

The component of SAS Decision Services that implements run-time services to create decisions synchronously or accepts requests for creating decisions asynchronously. Engines are generally clustered to meet availability and scalability requirements.

Decision Services Environment

A decision services environment offers services such as execution or run-time services, administrative services, and design-time services. These services are exposed as REST APIs. While most of these APIs are used to create, manipulate, and use artifacts in a single decision services environment, some APIs allow the transfer of artifacts from one environment to another.

Validity Rules

The artifacts in a decision services environment are complex objects that are created by design-time services. Parts of an artifact might depend on, or refer to, another part of the same artifact or another artifact. For example, a decision flow might contain a rule that uses a variable, also contained in the same decision flow. The variable might get its value from a binding in a decision definition that the decision flow refers to. In the latter case, the decision flow is considered to be dependent on the decision definition. Validity rules determine whether the contents of a single artifact is consistent in itself as well as across artifacts. In other words, in the above example, the variable that is referenced in the rule must not only exist in the decision flow, but must also be used in the rule that is consistent with its data type. Therefore, if the type of the variable is string, the rule might not use it as an integer. In addition, the binding in the decision definition that supplies the value to the variable must exist and match the type of the variable.

Administrative Services

A subset of the decision services that focuses on administering a decision services environment. In particular, it includes activating or de-activating decision flows, changing the time-out values, and changing the value of variables in the engines that are contained in the environment.

Activation

Activation makes the decision flows available for execution in a decision services engine or engines. Active decision flows can be executed to create decisions by sending decision requests to the engine. Not all decision flows can be activated. Only a decision flow that (by itself and the artifacts it depends on) satisfies validity rules can be activated. Since artifacts are modified through the design-time API, and moved to and from an environment using a promotion API. The administration API provides error messages that direct the user to one of these APIs in case validity errors are encountered during activation.

Run-time Services

A subset of the decision services that focuses on the execution of decision flows.

Decision

The output of the SAS Decision Services engine, generated by executing a decision flow. It includes a set of bindings, such as name-value pairs that contain the output that is generated by executing the decision flow. In addition, it contains diagnostic information, such as timestamps that mark the start and end of the decision-making process and a correlation ID that was supplied with the decision request.

Decision Request

The input to execute a decision flow in the SAS Decision Services engine. It also includes a set of bindings, such as name-value pairs that are input to the decision flow

in order to create the decision. In addition, it contains information, such as the Internet Assigned Numbers Authority (IANA) time zone of the client, that might be used to interpret clock time in the engine. It also contains the correlation ID for tracking the request through the engine. It can also contain the simulation timestamp and simulation time zone information. The latter two values are optional. However, they might be required by certain decision flows.

Decision Flow

A set of rules and logic that is executed in the SAS Decision Services engine when a decision request is received.

Decision Variable

The decision variables are named and typed values that influence the execution of decision flows in the SAS Decision Services engines. The value of a variable for a particular environment can be changed using the administration API for that environment. Variables are global in scope, for a particular environment. These variables affect the execution of all decision flows in the environment.

Decision Definition

Named metadata that describes the input and output bindings for the decision request and decision. The decision definition captures the external contract of the decision flow. It determines the names and types of input data that is expected by the flow, and that are to be supplied in a decision request. It also determines the names and types of data that is contained in the decision that the flow generates.

Bindings (input and output)

A set of name-value pairs that is used to represent data as part of the decision (output) or the decision request (input). The names and the type of corresponding value are defined in the decision definition. The name is usually chosen by the person who designs the decision definition, and it reflects the domain-specific terms. For example, it is possible to have a value called `customerId` of type `string`.

Decision ID

The name of the decision definition. The client application does not reference the decision flow directly. Instead, it uses the decision ID (the URL-encoded name of the decision definition) to request a decision to be created. This separates the client application from the actual execution logic, allowing the latter to be swapped out for another decision flow, without bringing the system down.

Time Out

Since, in most cases, it is critical to generate a decision within strict time limits, a decision definition can have a time-out value associated with it that causes the engine to time out if it fails to finish executing the decision flow within that time. The time-out values are specified in seconds in REST APIs.

Job

A collection that allows asynchronous execution of decision flows in the engine. The client application posts the decision request to this collection, and the engine queues a decision for execution using the decision request. As soon as the execution is queued, the engine returns a status of Accepted to the client application.

Pagination

Overview

Pagination establishes the baseline expected query parameters and behavior for paging or iterating through collections.

For collections, APIs can use query parameters to fetch (GET) or update (PUT, PATCH) subsets of collections. This helps prevent the transmission of entire (large) collections across networks, when a client application can visit or present only a small subset of the collection.

```

...collection?start=<item-index>&limit=<max-item-count>
...collection?start=<item-index>
...collection?limit=<max-item-count>
    
```

APIs should use the start value to indicate the starting index of the subset. Start should be a zero-based index. The default start should be 0.

APIs should use the limit value as the maximum number of items to return. The number of items returned can be less, if the collection is exhausted. APIs should use a default limit of 10.

Collection results should use hypermedia controls for the following, if each such collection subset or page navigation is possible and practical.

- first (rel="first")
- previous (rel="prev")
- next (rel="next")
- last (rel="last")

The query result uses the collection format described in [“application/vnd.sas.collection” on page 254](#). The collection result must omit the rel="next" link if the collection has been exhausted. This means that there are no more items. If the pagination is at the beginning of the collection, the API must omit the rel="prev" link. The collection object can contain an integer count attribute, if the collection size is known.

Guidance

When a resource representation contains links to collections, those links should go to the main collection without the query parameters. The server should apply the default start and limit values to return the first page. The API can also include a field that contains the number of items in the collection. By convention, this is named itemCount for the corresponding .../items collection. If you want to view all of the objects within the collection, use at least the total number of objects as the value of limit. Pagination should be the default, to prevent the possibility of reading large collections.

Requests for collection resources return an empty collection, if there are no items that match the criteria. For example, if the start value is more than the number of items in the collection, an empty collection is returned. This could happen if resources are

deleted while someone is paging through the collection using the next or previous links that were constructed when the resources were still available. This is not an error condition and does not generate an error response.

Media Types

Externally Defined Media Types

application/vnd.sas.collection

`application/vnd.sas.collection` denotes a collection of resources. The collection should contain domain objects in the element named `items`. This allows heterogeneous collection and uniform structure across different collections. It should have a `name` attribute that is the domain-specific name for the collection items. The default is `"items"`. If approved by the API COE, an API can use a different name for the container. The value of the `accept` attribute, if present, must be a text string consisting of a space-separated list of media type strings. The media type names can omit the `+xml` and `+json`.

The type is implicit in the response representation of the collection. It should contain the values of the `start` and `index` of the collection's subset of elements, as per the pagination query parameters (`&start=int` and `&limit=int`), or the collection's default `start` (0) and `limit` (10). The type of `start` is a 64-bit signed integer value. The type of `limit` is a 32-bit signed integer value. The collection can include an integer count attribute if the API can efficiently compute the size of the underlying collection. The type of the count is a 64-bit signed integer value. The collection can have a nested collection of links. Each link element in the links collection must be represented according to REST API link representation standards.

Here is an example of `application/vnd.sas.collection+json` and `application/vnd.sas.collection+json;version=2`:

```
{
  "version" : 2,
  "accept": "space-separated media type names allowed in this collection",
  "count" : integer,
  "start" : integer,
```



```

"limit" : integer,
"name" : "items",
"items": [
  { resource1 fields }, ...,
  { resourceN fields }
],
"links" : [
  { link representation }, ...
  { link representation },
]
}

```

Note: The order of the fields can vary.

Here is an example of `application/vnd.sas.collection+xml` and `application/vnd.sas.collection+xml;version=2`

```

<collection version="2"
  name="items"
  accept="space-separated media type names allowed in this collection"
  start="integer"
  limit="integer"
  count="integer">
  <items>
    <resource1 xml/>
    <resourceN xml/>
  </items>
  <links>
    <link>...</link>
    ...
    <link>...</link>
  </links>
</collection>

```

Note: The order of the fields can vary.

SAS Decision Services Media Types

application/vnd.sas.decision.definition

The `application/vnd.sas.decision.definition` media type describes the inputs that are required to execute a decision flow and outputs that are generated by it. It describes the data that can be used to construct the input member of the decision request and render the data that is contained in the output member of the created decision. The value of the

decisionId field is used as a key to execute decision flows synchronously, as well as asynchronously. When returned by the run-time services, it represents a decision definition of a decision flow deployed in the engine. When returned by the design services, it represents a decision definition that can be edited and used to construct decision flow. The latter case is not addressed by this API.

Here are the link relations for the application/vnd.sas.decision.definition media type.

Relationship	HTTP Method	Description
self	GET	<p>The link to the summary of this decision definition.</p> <p>URI: SASDecisionServices/rest/runtime/decisionDefinitions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.definition.summary</p>
self	GET	<p>The link to itself.</p> <p>URI: SASDecisionServices/rest/runtime/decisionDefinitions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.definition</p>
create	POST	<p>The link to create a decision by executing the decision flow associated with the underlying decision definition.</p> <p>URI: SASDecisionServices/rest/runtime/decisions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.request</p>

Relationship	HTTP Method	Description
execute	POST	<p>The link to create a job that executes the decision flow that is associated with the underlying decision definition. Use this to execute a decision flow asynchronously (for example without waiting for it to finish).</p> <p>URI: SASDecisionServices/rest/runtime/jobs?decisionId={decisionId}</p> <p>Media type: application/vnd.sas.decision.request</p>

The application/vnd.sas.decision.definition media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
decisionID	string	The ID or name of the underlying decision definition.
description	string	Short description of the decision definition as entered by the creator of the resource.
created	dateTime	The timestamp that shows when this resource was first created.
modified	dateTime	The timestamp of when this resource was last modified.

Name	Type	Description
inputs	object	<p>This is a set of bindings such as an unordered set of name-value pairs that describe the data that is accepted by the decision flow. It can be used to construct the input member of the decision request when executing a flow. The name part of the binding is a string that represents the name of a field. The value is a string that is the type of data the decision request would contain for that field. The valid types are described below. While the decision definition contains only the name and type of a field, the description below also addresses how the data of the corresponding type would look as part of the decision request.</p> <p>SAS Decision Services supports data values of the following primary types:</p> <ul style="list-style-type: none"> ■ Boolean — This is a Boolean value that includes true, false, or null. ■ integer — This is a 32-bit signed two's complement integer value including null representing a missing value. ■ decimal — This is a double-precision 64-bit format IEEE 754 value including "Infinity", "-Infinity", * "NaN", and null representations. ■ string — This is a character string that can be empty or null. ■ dateTime — This is a timestamp value, with the accuracy of milliseconds that can be null. In JSON, this is represented as a string in ISO8601 format. In XML, it is represented as the dateTime type. <p>Single dimensional arrays or lists of the above types are also supported. Here are the corresponding types:</p> <ul style="list-style-type: none"> ■ booleanArray ■ integerArray ■ decimalArray ■ stringArray ■ dateTimeArray

Name	Type	Description
inputs (continued)	object	<p>A table type is also supported. A table is a dynamic type that contains self-describing metadata determining the name and type of data for each column. Table data is represented as an array of two objects: the metadata object and the data object. The metadata element must precede the data element because it is often easier to parse the metadata before the data. Therefore, the parsing code is smaller and simpler for the client. The objects are described below:</p> <p>metadata Contains an ordered collection of column metadata that includes the name and a string representing the type of the column, as described above. The type of the column can be one of the primary types described above. Array types and nested tables are not supported.</p> <p>data Contains an ordered collection of values representing the data values of the table, in row major order. For example, the outer collection represents a collection of rows and the inner collection represents a single row of the table. The values in each row must match the type of the columns in the order described in the metadata object above.</p>
outputs	object	This is a set of bindings such as an unordered set of name-value pairs that describe the data returned by the decision flow. It can be used to interpret the output member of the decision when executing a flow. The supported types are identical to those described in the inputs section. See the inputs member above for details.
links	collection of link objects	One or more link objects. See the link relations information above for a description of the link types.

Here is an example of `application/vnd.sas.decision.definition+json`:

```
{
  "decisionId": "Sample Offers",
  "description": "This is a sample decision definition summary.",
  "created" : "2014-01-27T12:32:12.345Z",
  "modified" : "2014-01-27T12:32:12.345Z",
  "inputs" : {
    "aBoolean" : "boolean",
    "anInt" : "integer",
```

```

    "aFloat" : "decimal",
    "aString" : "string",
    "aDateTime" : "dateTime",
    "aBooleanArray" : "booleanArray",
    "aIntArray" : "integerArray",
    "aFloatArray" : "decimalArray",
    "aStringArray" : "stringArray",
    "aTimestampArray" : "dateTimeArray"
    "aTable" : "table"
  },
  "outputs" : {
    "aBoolean" : "boolean",
    "anInt" : "integer",
    "aFloat" : "decimal",
    "aString" : "string",
    "aDateTime" : "dateTime",
    "aBooleanArray" : "booleanArray",
    "aIntArray" : "integerArray",
    "aFloatArray" : "decimalArray",
    "aStringArray" : "stringArray",
    "aTimestampArray" : "dateTimeArray"
    "aTable" : "table"
  },
  "links": [{
    "method": "GET",
    "rel": "self",
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/
      runtime/decisionDefinitions/Sample%20Offers",
    "uri": "decisionDefinitions/Sample%20Offers",
    "type": "application/vnd.sas.decision.definition"
  }, {
    "method": "GET",
    "rel": "summary",
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/
      runtime/decisionDefinitions/Sample%20Offers",
    "uri": "decisionDefinitions/Sample%20Offers",
    "type": "application/vnd.sas.decision.definition.summary"
  }, {
    "method": "POST",
    "rel": "decisions",
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/
      runtime/decisions/Sample%20Offers",
    "uri": "decisions/Sample%20Offers",
    "type": "application/vnd.sas.decision"
  }, {
    "method": "POST",
    "rel": "jobs",

```

```
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/  
        runtime/jobs/Sample%20Offers",  
    "uri": "jobs/Sample%20Offers"  
  }  
}
```

application/vnd.sas.decision.definition.summary

Summary information for the underlying decision definition that is returned as part of a collection to support browse functionality that does not require retrieving the entire decision definition. This information might be returned by the run-time services. In that case, the underlying resource has been deployed and is not editable.

Here are the link relations for the application/vnd.sas.decision.definition.summary media type.

Relationship	HTTP Method	Description
self	GET	<p>The link to the summary of this decision definition.</p> <p>URI: SASDecisionServices/rest/runtime/decisionDefinitions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.definition.summary</p>
self	GET	<p>The link to itself.</p> <p>URI: SASDecisionServices/rest/runtime/decisionDefinitions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.definition</p>

Relationship	HTTP Method	Description
create	POST	<p>The link to create a decision by executing the decision flow that is associated with the underlying decision definition.</p> <p>URI: SASDecisionServices/rest/runtime/decisions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.request</p>
execute	POST	<p>The link to create a job that executes the decision flow that is associated with the underlying decision definition. Use this to execute a decision flow asynchronously (for example, without waiting for it to finish).</p> <p>URI: SASDecisionServices/rest/runtime/jobs?decisionId={decisionId}</p> <p>Media type: application/vnd.sas.decision.request</p>

The application/vnd.sas.decision.definition.summary media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
decisionID	string	The ID or name of the underlying decision definition.
description	string	Short description of the decision definition as entered by the creator of the resource.

Name	Type	Description
created	dateTime	The timestamp of when this resource was first created.
modified	dateTime	The timestamp that shows when this resource was last modified.
links	collection of link objects	One or more link objects. See the link relations information above for a description of the link types.

Here is an example of `application/vnd.sas.decision.definition.summary+json`:

```
{
  "decisionId": "Sample Offers",
  "description": "This is a sample decision definition summary.",
  "created" : "2014-01-27T12:32:12.345Z",
  "modified" : "2014-01-27T12:32:12.345Z",
  "links": [{
    "method": "GET",
    "rel": "self",
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/
      runtime/decisionDefinitions/Sample%20Offers",
    "uri": "decisionDefinitions/Sample%20Offers",
    "type": "application/vnd.sas.decision.definition.summary"
  }, {
    "method": "GET",
    "rel": "detail",
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/
      runtime/decisionDefinitions/Sample%20Offers",
    "uri": "decisionDefinitions/Sample%20Offers",
    "type": "application/vnd.sas.decision.definition"
  }, {
    "method": "POST",
    "rel": "decisions",
    "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/
      runtime/decisions/Sample%20Offers",
    "uri": "decisions/Sample%20Offers",
    "type": "application/vnd.sas.decision"
  }, {
    "method": "POST",
    "rel": "jobs",
```

```
        "href": "http://rdcesx13020.race.sas.com/SASDecisionServices/rest/  
                runtime/jobs/Sample%20Offers",  
        "uri": "jobs/Sample%20Offers"  
    }  
}
```

application/vnd.sas.decision

The application/vnd.sas.decision media type describes the decision as returned by the SAS Decision Services engine when a decision request is posted to the decisions resource collection.

The application/vnd.sas.decision media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
correlationId	string	A string value that is supplied through the decision request and that can be used to correlate the request and the decision that is generated. The engine also uses this string in the log to identify errors or execution trace statements that are associated with the generation of the decision object.
startTimestamp	dateTime	Server timestamp that shows when the engine started generating the decision.
endTimestamp	dateTime	Server timestamp that shows when the engine completed generating this decision.

Name	Type	Description
outputs	object	This is a set of bindings such as an unordered set of name-value pairs that are a part of the decision. The names correspond to the binding names that are described in the outputs member of decision definition. The value contains data of the corresponding type. The valid values are described in the inputs member of application/vnd.sas.decision.definition.

Here is an example of application/vnd.sas.decision+json:

```
{
  "correlationId" : "ABCD",
  "startTimestamp" : "2008-11-20T12:40:53.007-05:00",
  "endTimestamp" : "2008-11-20T12:40:53.007-05:00",
  "outputs" : {
    "aBoolean" : true,
    "anInt" : 111,
    "aFloat" : 1.11,
    "aString" : "String1",
    "aDateTime" : "2007-07-13T14:32:07.000Z",
    "aNullValue" : null,
    "aBooleanArray" : [ true, false, true ],
    "aIntArray" : [ 111, 222, 333 ],
    "aFloatArray" : [ 1.11, 2.22, 3.33 ],
    "aStringArray" : [ "String1", "String2", "String3" ],
    "aTimestampArray" : [ "2007-07-13T14:32:07.000Z",
                          "2007-02-13T14:32:07.000Z",
                          "2007-02-13T00:00:00.000Z" ],
    "aBooleanArrayWithANullItem" : [ true, false, null ],
    "aTable" : [{
      "metadata" : [
        { "aStringColumn" : "string" },
        { "anIntColumn" : "int" },
        { "aFloatColumn" : "float" },
        { "aBooleanColumn" : "boolean" },
        { "aTimestampColumn" : "timestamp" }
      ]
    }], {
    "data" : [ [ "one", 1, 1.11, true, "2001-01-01T01:01:01.000Z" ],
```

```
        [ "two", 2, 2.22, false, "2002-02-02T02:02:02.000Z" ],
        [ "three", 3, 3.33, true, "2003-03-03T03:03:03.000Z" ],
        [ "four", 4, 4.44, false, "2004-04-04T04:04:04.000Z" ],
        [ "five", 5, 5.55, true, "2005-05-05T05:05:05.000Z" ],
    [ null, null, null, null, null ]]
  ]],
  "aTableWithNoData" : [{
    "metadata" : [
      { "aStringColumn" : "string" },
      { "anIntColumn" : "int" },
      { "aFloatColumn" : "float" },
      { "aBooleanColumn" : "boolean" },
      { "aTimestampColumn" : "timestamp" }]
    }, {
      "data" : []
    }
  ]],
  "aNullValuedTable" : null
}
```

application/vnd.sas.decision.request

The application/vnd.sas.decision.request media type represents the information that is used to generate the decision.

The application/vnd.sas.decision.request media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
correlationId	string	A string value that is supplied through the decision request and that can be used to correlate the request and the decision that is generated. The engine also uses this string in the log to identify errors or execution trace statements that are associated with the generation of this decision object. This value is same as in the vnd.sas.decision media type.

Name	Type	Description
clientTimeZone	string	A string that contains the time zone of the client that requests the decision, as defined in the IANA time zone database. This value can be used by the decision flow to evaluate datetime values in the client's time zone for the purpose of generating a decision.
inputs	object	This is a set of bindings such as an unordered set of name-value pairs. The names correspond to the binding names that are described in the inputs member of the decision definition. The value contains the data of the corresponding type. The valid values are described in the outputs member of application/vnd.sas.decision.definition.

Here is an example of application/vnd.sas.decision.request+json:

```
{
  "correlationId" : "ABCD",
  "clientTimeZone" : "Asia/Kolkata",
  "inputs" : {
    "aBoolean" : true,
    "anInt" : 111,
    "aFloat" : 1.11,
    "aString" : "String1",
    "aDateTime" : "2007-07-13T14:32:07.000Z",
    "aNullValue" : null,
    "aBooleanArray" : [ true, false, true ],
    "aIntArray" : [ 111, 222, 333 ],
    "aFloatArray" : [ 1.11, 2.22, 3.33 ],
    "aStringArray" : [ "String1", "String2", "String3" ],
    "aTimestampArray" : [ "2007-07-13T14:32:07.000Z",
      "2007-02-13T14:32:07.000Z", "2007-02-13T00:00:00.000Z" ]
    "aBooleanArrayWithANullItem" : [ true, false, null ],
  }
}
```

```

    "aTable" : [{
      "metadata" : [
        { "aStringColumn" : "string" },
        { "anIntColumn" : "int" },
        { "aFloatColumn" : "float" },
        { "aBooleanColumn" : "boolean" },
        { "aTimestampColumn" : "timestamp" }]
    }, {
      "data" : [[ "one", 1, 1.11, true, "2001-01-01T01:01:01.000Z" ],
        [ "two", 2, 2.22, false, "2002-02-02T02:02:02.000Z" ],
        [ "three", 3, 3.33, true, "2003-03-03T03:03:03.000Z" ],
        [ "four", 4, 4.44, false, "2004-04-04T04:04:04.000Z" ],
        [ "five", 5, 5.55, true, "2005-05-05T05:05:05.000Z" ],
        [ null, null, null, null, null ]]
    }],
    "aTableWithNoData" : [{
      "metadata" : [
        { "aStringColumn" : "string" },
        { "anIntColumn" : "int" },
        { "aFloatColumn" : "float" },
        { "aBooleanColumn" : "boolean" },
        { "aTimestampColumn" : "timestamp" }]
    }, {
      "data" : []
    }],
    "aNullValuedTable" : null
  }
}

```

Resources

Collection /

The / returns a collection of links to the top-level collections surfaced through this API.

Authentication is not required. The request URL is GET <http://www.example.com/SASDecisionServices/rest/runtime/>. The response to the GET request is a collection of top-level links that are returned to decisionDefinitions for resources that support run-time services. Currently only a single collection is returned.

Here are the HTTP response codes:

200
OK

404
Not found.

500
Server error.

This resource can return the following media type representations by setting the Accept header of the request:

- application/json
- application/vnd.sas.collection+json

Collection /decisionDefinitions

The /decisionDefinitions collection is a collection of decision definitions that are associated with the corresponding decision flows that are deployed in the SAS Decision Services engine. The collection supports pagination and filtering by the name of the decision definition. Because the names of decision definitions that are deployed in a SAS Decision Services engine are unique, using such a filter returns a collection of at most one item. The decision definition summary objects contain links that allow the execution of the decision flow synchronously and asynchronously. It is also possible to retrieve the complete decision definition. Authentication is not required.

The GET request URL is GET <http://www.example.com/SASDecisionServices/rest/runtime/decisionDefinitions>.

Here are the HTTP response codes:

200
OK

404
Not found.

500
Server error.

Here are the query parameters for /decisionDefinitions:

Name	Type	Description
start	integer	The starting index of the first decision definition summary on a page. The index is 0-based. The default is 0.
limit	integer	The maximum number of decision definition summary objects to return on this page of results. The actual number of returned decision definition summary objects might be less, if the collection has been exhausted. The default is 10.
name	string	This filters by the name of the decision definition. The names of decision definitions are unique. Therefore, this returns a collection of at most one item.

Here is a JSON representation of the decisionDefinitionSummaries collection:

```
{ "decisionDefinitionSummaries":
  [
    { application/vnd.sas.decision.definition.summary contenti },
    { application/vnd.sas.decision.definition.summary contenti+1 },
    ...
    { application/vnd.sas.decision.definition.summary contentn },
  ],
  "accept": "application/vnd.sas.decision.definition.summary"
  "links": [
    { "rel" : "first", "method" : "GET", ... },
    { "rel" : "prev", "method" : "GET", ... },
    { "rel" : "next", "method" : "GET", ... },
    { "rel" : "last", "method" : "GET", ... }
  ]
}
```


This resource can return the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.collection+json

Collection /jobs

The /jobs collection is a collection of resources that represent transient requests to asynchronously execute decision flows. When a decision.request object is posted to this URL along with a URL parameter that contains the decisionId of the decision definition that is associated with a decision flow, the SAS Decision Services engine accepts this request and returns a status of 202 Accepted. It then creates a job that represents a request to execute the decision flow asynchronously. The client application does not wait for the flow to execute or even for the job to be created. Because a job is a transient resource, it is not possible to retrieve a job object using a GET method. Authentication is not required.

The POST URL is POST <http://www.example.com/SASDecisionServices/rest/runtime/jobs>.

Here are the HTTP response codes:

200

OK

404

Not found.

500

Server error.

Here is the query parameter for /jobs:

Name	Type	Description
decisionID	string	The ID of the decision definition that is associated with the decision flow to be executed asynchronously.

This method can return the following content type, as named by the Content-Type: header:

- application/vnd.sas.decision.request+json

Resource /decisionDefinitions/{decisionID}

The /decisionDefinitions/{decisionId} resource represents a single decision definition that is associated with a decision flow that is deployed in the SAS Decision Services engine. The GET request returns a single decision definition as identified by either the decisionId or the corresponding decision definition summary object, depending on the media type that is requested.

The GET request URL is GET <http://www.example.com/SASDecisionServices/rest/runtime/decisionDefinitions/{decisionId}>. Authentication is not required.

Here are the HTTP response codes:

- 200
OK
- 404
Not found.
- 500
Server error.

This resource can return the following media type representations by setting the Accept: header of the request:

- application/json
- application/vnd.sas.decision.definition+json

■ application/vnd.sas.decision.definition.summary+json

Here is the query parameter for /jobs:

Name	Type	Description
decisionID	string	The URL encoded name of the decision definition.

Resource /decisions/{decisionId}

The /decisions/{decisionId} resource is a transient resource that is used to execute decision flows in the SAS Decision Services engine. The decision resource is created when a decision.request object is posted at the URL below. The resource that is created is not persisted and cannot be retrieved using a GET method. Therefore, after generation by using a separate GET method, decisions cannot be accessed. The POST method returns the decision object in the response body and does not return a location for the created decisions. It executes a decision flow that is associated with a decision definition that is identified by the decisionId. Authentication is not required.

The POST URL is POST <http://www.example.com/SASDecisionServices/rest/runtime/decisions/{decisionId}>.

Here are the HTTP response codes:

200

OK

404

Not found.

500

Server error.

This method can return the following content type, as named by the Content-Type: header:

■ application/vnd.sas.decision.request+json

This resource can return the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.decision.definition+json

Post-Installation Steps

In order for the REST API to work correctly, the following post-installation steps must be completed:

- 1 Find the sas.conf file in the conf folder of the SAS web server installation <SAS-configuration-directory>\Web\WebServer\conf.
- 2 Find the two lines of code that define the proxy and reverse proxy for the SAS Decision Services engine.

```
ProxyPass /RTDM balancer://<machine name>.na.sas.com_Cluster7/RTDM
ProxyPassReverse /RTDM balancer://<machine name>.na.sas.com_Cluster7/RTDM
```

- 3 Copy these lines of code to a text editor and change the /RTDM on the right side of the mapping to /SASDecisionServices as follows:

```
ProxyPass /SASDecisionServices balancer://<machine name>.na.sas.com_Cluster7/RTDM
ProxyPassReverse /SASDecisionServices balancer://<machine name>.na.sas.com_Cluster7/RTDM
```

- 4 Paste these modified lines of code directly below the lines mapping /RTDM.

```
ProxyPass /RTDM balancer://<machine name>.na.sas.com_Cluster7/RTDM
ProxyPassReverse /RTDM balancer://<machine name>.na.sas.com_Cluster7/RTDM
ProxyPass /SASDecisionServices balancer://<machine name>.na.sas.com_Cluster7/RTDM
ProxyPassReverse /SASDecisionServices balancer://<machine name>.na.sas.com_Cluster7/RTDM
```

- 5 After editing the file, restart the SAS web server.

Appendix 6

SAS Decision Services Administration API

- Overview* 275
- Use Cases* 276
- Terminology* 277
- Media Types* 277
 - Externally Defined Media Types 277
 - SAS Decision Services Media Types 277
- Resources and Collections* 295
 - Resource / 295
 - Collection /decisionDefinitions 297
 - Collection /decisionFlows 307
 - Collection /variables 315

Overview

The SAS Decision Services Administration API is a REST API that administers a SAS Decision Services deployment containing one or more SAS Decision Services engines. Decision flows that are loaded in the engine for execution are known as *active decision flows*. The process of making a decision flow active is known as *activation*. Conversely, making it unavailable is known as *deactivation*. Not all decision flows can be activated.

Only decision flows that are valid can be activated. The validity rules are described in the terminology section. The Administration API allows activation of valid decision flows and deactivation of any active decision flow.

The Administration API does not support modifying artifacts such as decision flows, or adding missing artifacts from other decision services deployments. So, if activation fails because of validity reasons, the administration API returns detailed validation error messages. These messages can then be used to correct the situation by using the appropriate API.

In most cases in the engine, the decision must be generated under strict latency constraints. Every decision flow in the engine is associated with a decision definition that contains a timeout value. This timeout value is used by the engine to terminate the decision flow execution, should it take more time to generate the decision. The administrative API allows setting and changing the timeout value of the decision definition.

Some decision flows can be associated with a variable whose value can affect the execution of the decision flow and, accordingly, the decision generated by it. The Administration API also makes it possible to change the value of variables.

Use Cases

The following are possible use cases for the SAS Decision Services Administration API:

Query the status of flows in the engine

You might want to select a group of flows based on certain filter criteria and view their contents. The filter criteria might include selection by created or modified date, name, ID, activation status, association with a particular decision definition, association with a particular variable, or a combination of those. Decision flows in general are large resources and, for the purpose of administration, a summary representation is more helpful.

Activation or deactivation

You might want to select a group of inactive flows based on filter criteria, as above, and activate or deactivate one or all of them.

Changing timeout values

You might want to select a decision definition and view or change its timeout value. Again, a summary representation of the decision definition is more useful.

Changing the value of a variable

You might want to select a variable using filter criteria and view or change its value. The filter criteria might include an association with a particular flow.

Terminology

For terminology specific to REST APIs, see [“Terminology” on page 248](#).

Media Types

Externally Defined Media Types

application/vnd.sas.collection

For more information about this media type, see [“application/vnd.sas.collection” on page 254](#).

SAS Decision Services Media Types

application/vnd.sas.decision.definition.summary

Summary information for the underlying decision definition is returned as part of a collection. This makes it possible to support browse functionality that does not require retrieving the entire decision definition. This information might be returned by the run-time services. In that case, the underlying resource has been deployed and is not editable. When exposed by the design-time services links, it includes those that help create, edit, and delete decision definition resources. When returned by the administration services, it represents a decision definition that is available in the engine that is associated with a decision flow that might or might not be active.

Here are the link relations for the application/vnd.sas.decision.definition.summary media type.

Relationship	HTTP Method	Description
self	GET	<p>The link to the summary of this decision definition. Available in the Execution API.</p> <p>URI: /rest/decisionDefinitions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.definition.summary</p>
self	GET	<p>The link to itself. Available in the Execution API and Administration API.</p> <p>URI: /rest/decisionDefinitions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.definition</p>
create	POST	<p>The link to create a decision by executing the decision flow that is associated with the underlying decision definition. Available in the Execution API and Administration API.</p> <p>URI: /rest/decisions/{decisionId}</p> <p>Media type: application/vnd.sas.decision.request</p>

Relationship	HTTP Method	Description
execute	POST	<p>The link to create a job that executes the decision flow that is associated with the underlying decision definition. Use this to execute a decision flow asynchronously (for example, without waiting for it to finish). Available in the Execution API and Administration API.</p> <p>URI: SASDecisionServices/rest/runtime/jobs?decisionId={decisionId}</p> <p>Media type: application/vnd.sas.decision.request</p>
timeOut	GET	<p>The link to return a floating point number for the timeout, in seconds, for this decision definition. If the value is not set, an empty document is returned. If the API does not support this value, a Not Found or 404 error is returned. Available in the Administration API.</p> <p>URI: /rest/decisionDefinitions/{decisionId}/timeOut</p> <p>Media type: text/plain</p>

Relationship	HTTP Method	Description
timeOut	PUT	<p>The link to update the timeout, in seconds, for this decision definition. Acceptable values are a positive floating point number or an empty document. The latter clears the value. If the API does not support this value, a Not Found or 404 error is returned. Available in the Administration API.</p> <p>URI: /rest/decisionDefinitions/{decisionId}/timeOut</p> <p>Media type: text/plain</p>
timeOutEnabled	GET	<p>The link to return a Boolean expression indicating whether the timeout is enabled for the decision definition. If the value is not set, an empty document is returned. If the API does not support this value, a Not Found or 404 error is returned. Available in the Administration API.</p> <p>URI: /rest/decisionDefinitions/{decisionId}/timeOutEnabled</p> <p>Media type: text/plain</p>

Relationship	HTTP Method	Description
timeOutEnabled	PUT	<p>The link to update the timeOutEnabled field of this decision definition. Acceptable values are a Boolean expression or an empty document. The latter clears the value. If the API does not support this value, a Not Found or 404 error is returned. Available in the Administration API.</p> <p>URI: /rest/decisionDefinitions/{decisionId}/timeOutEnabled</p> <p>Media type: text/plain</p>

The application/vnd.sas.decision.definition.summary media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
id	string	The system-assigned unique ID for this object.
decisionID	string	The ID or name of the underlying decision definition.
label	string	The descriptive or display name of the underlying decision definition.
description	string	The short description of the decision definition as entered by the creator of the resource.

Name	Type	Description
created	dateTime	The timestamp that shows when this resource was first created.
modified	dateTime	The timestamp that shows when this resource was last modified.
lastModifiedBy	string	The name of the user who last modified the resource.
timeOutEnabled	Boolean	It is true, if the timeout for this decision definition is enabled. It is false, if otherwise.
timeOut	decimal	The value of the timeout in seconds.
links	collection of link objects	One or more link objects. See the link relations information above for a description of the link types.

Here is an example of application/vnd.sas.decision.definition.summary+json:

```
{
  "version": 1,
  "decisionId": "Sample Offers",
  "description": "This is a sample decision definition summary.",
  "created" : "2014-01-27T12:32:12.345Z",
  "modified" : "2014-01-27T12:32:12.345Z",
  "links": [{
    "method": "GET",
    "rel": "self",
    "href": "http://<server>/SASDecisionServices/
              rest/runtime/decisionDefinitions/Sample%20Offers",
    "uri": "decisionDefinitions/Sample%20Offers",
    "type": "application/vnd.sas.decision.definition.summary"
  }, {
    "method": "GET",
    "rel": "detail",
    "href": "http://<server>/SASDecisionServices/
              rest/runtime/decisionDefinitions/Sample%20Offers",
```

```
    "uri": "decisionDefinitions/Sample%20Offers",
    "type": "application/vnd.sas.decision.definition"
  }, {
    "method": "POST",
    "rel": "decisions",
    "href": "http://<server>/SASDecisionServices/
             rest/runtime/decisions/Sample%20Offers",
    "uri": "decisions/Sample%20Offers",
    "type": "application/vnd.sas.decision"
  }, {
    "method": "POST",
    "rel": "jobs",
    "href": "http://<server>/SASDecisionServices/
             rest/runtime/jobs/Sample%20Offers",
    "uri": "jobs/Sample%20Offers"
  }
}
```

application/vnd.sas.decision.flow.summary

The application/vnd.sas.decision.flow.summary media type is a summary representation of a decision flow.

Here are the link relations for the application/vnd.sas.decision.flow.summary media type.

Relationship	HTTP Method	Description
self	GET	The link to itself. Available in the Administration API. URI: /rest/decisionFlows/{decision flow id} Media type: application/vnd.sas.decision.flow.summary

Relationship	HTTP Method	Description
decisionDefinition	GET	<p>The link to the decision definition of this decision flow. Available in the Administration API.</p> <p>URI: /rest/decisionDefinitions/{decision id}</p> <p>Media type: application/vnd.sas.decision.definition.summary</p>
active	GET	<p>The link to return a Boolean expression that indicates whether the decision flow is active. If the API does not support this value, a Not Found or 404 error is returned. Available in the Administration API.</p> <p>URI: /rest/decisionFlows/{decision flow id}/active</p> <p>Media type: text/plain</p>
active	PUT	<p>The link to update the active field of this decision flow. Acceptable values are a Boolean expression. If the API does not support this value, a Not Found or 404 error is returned. Available in the Administration API.</p> <p>URI: rest/decisionFlows/{decision flow id}/active</p> <p>Media type: text/plain</p>

The application/vnd.sas.decision.flow.summary media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
id	string	The system-assigned unique ID for this object.
label	string	The descriptive or display name of the underlying decision flow.
description	string	The short description of the decision flow, as entered by the creator of the resource.
created	dateTime	The timestamp that shows when this resource was last modified.
modified	dateTime	The timestamp that shows when this resource was last modified.
lastModifiedBy	string	The name of the user who last modified the resource.
decisionID	string	The ID or name of the associated decision definition.
active	Boolean	This is true if the flow has been activated. Otherwise, this is false.
links	collection of link objects	One or more link objects. See the link relations information above for a description of the link types.

Here is an example of application/vnd.sas.decision.flow.summary:

```
{
  "version": 1,
```

```

    "id": "AM4OPKD4J0HCIAAA",
    "description": "ExitPopupNoUCH:PerformanceTestFlows:CMDM Performance:",
    "label" : "ExitPopupNoUCH:PerformanceTestFlows:CMDM Performance"
    "created" : "2008-11-18T15:25:22.535+00:00",
    "modified" : "2008-11-18T15:25:22.535+00:00",
    "lastModifiedBy" : "prasen",
    "decisionId" : "ExitPopupNoUCHEvent",
    "active" : "true",
    "links": [{
        "method": "GET",
        "rel": "self",
        "href": "http://<server>/SASDecisionServicesAdministration/
                rest/decisionFlows/AM4OPKD4J0HCIAAA",
        "uri": "decisionFlows/AM4OPKD4J0HCIAAA",
        "type": "application/vnd.sas.decision.flow.summary"
    }, {
        "method": "GET",
        "rel": "decisionDefinition",
        "href": "http://<server>/SASDecisionServicesAdministration/
                rest/decisionDefinitions/ExitPopupNoUCHEvent",
        "uri": "decisionDefinitions/ExitPopupNoUCHEvent",
        "type": "application/vnd.sas.decision.definition.summary"
    }, {
        "method": "GET",
        "rel": "active",
        "href": "http://<server>/SASDecisionServicesAdministration/
                rest/decisionFlows/AM4OPKD4J0HCIAAA/active",
        "uri": "decisionFlows/AM4OPKD4J0HCIAAA/active",
        "type": "text/plain"
    }, {
        "method": "PUT",
        "rel": "active",
        "href": "http://<server>/SASDecisionServicesAdministration/
                rest/decisionFlows/AM4OPKD4J0HCIAAA/active",
        "uri": "decisionFlows/AM4OPKD4J0HCIAAA/active",
        "type": "text/plain"
    }
  ]
}

```

application/vnd.sas.decision.variable

The application/vnd.sas.decision.variable media type represents a variable in SAS Decision Services.

Here are the link relations for the application/vnd.sas.decision.variable media type.

Relationship	HTTP Method	Description
self	GET	<p>The link to itself. Available in the Administration API.</p> <p>URI: /rest/variables/{variable id}</p> <p>Media type: application/vnd.sas.decision.variable</p>
value	GET	<p>The link to return a representation for the value field of the variable. The representation of the value is described in the member's information below. The value is represented as a collection of strings that contain literal expressions. For array type, each value corresponds to the value that is assigned to that element of the array. Non-array variables use only the first element. Available in the Administration API.</p> <p>URI: /rest/variables/{variable id}/value</p> <p>Media type: application/vnd.sas.collection+json, application/vnd.sas.collection+xml</p>
value	PUT	<p>The link to update the value field of the variable. . Available in the Administration API.</p> <p>URI: /rest/variables/{variable id}/value</p> <p>Media type: application/vnd.sas.collection+json, application/vnd.sas.collection+xml</p>

The application/vnd.sas.decision.variable media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
id	string	The system-assigned unique ID for this object.
label	string	The descriptive or display name of the underlying variable.
description	string	The short description of the variable, as entered by the creator of the resource.
created	dateTime	The timestamp that shows when this resource was last modified.
modified	dateTime	The timestamp that shows when this resource was last modified.
lastModifiedBy	string	The name of the user who last modified the resource.
type	string	The type of variable.
links	collection of link objects	One or more link objects. See the link relations information above for a description of the link types.
value	array of strings	The value of the underlying variable as an array of strings. Each element corresponds to a literal expression that is evaluated to get the value of that element. Non-array variables require a single element. Array variables can use more than one element.

Here is an example of application/vnd.sas.decision.variable:

```
{
  "version": 1,
  "id": "GVStringArray1",
  "description": "A global variable holding an array of strings",
  "label" : "Global Variable String Array"
  "created" : "2008-11-18T15:25:22.535+00:00",
  "modified" : "2008-11-18T15:25:22.535+00:00",
  "lastModifiedBy" : "prasen",
  "type" : "stringArray",
  "value" : ["\"String1\"", "\"String2\"", "\"String3\""],
  "links": [{
    "method": "GET",
    "rel": "self",
    "href": "http://<server>/SASDecisionServicesAdministration/
              rest/variables/GVStringArray1",
    "uri": "variables/GVStringArray1",
    "type": "application/vnd.sas.decision.variable"
  }, {
    "method": "GET",
    "rel": "value",
    "href": "http://<server>/SASDecisionServicesAdministration/
              rest/variables/GVStringArray1/value",
    "uri": "variables/GVStringArray1/value",
    "type": "application/json"
  }, {
    "method": "PUT",
    "rel": "value",
    "href": "http://<server>/SASDecisionServicesAdministration/
              rest/variables/GVStringArray1/value",
    "uri": "variables/GVStringArray1/value",
    "type": "application/json"
  }]
}
```

application/vnd.sas.decision.batch.update.request

The application/vnd.sas.decision.batch.update.request media type represents a command to update multiple objects in a collection, identified by the IDs in the selection field. This is used for activating or deactivating multiple decision flows and to set the timeout value for multiple decision definitions in the SAS Decision Services Administration API.

The application/vnd.sas.decision.batch.update.request media type contains the following members.

Name	Type	Description
version	integer	The media type's schema version number.
selection	array	An array of IDs of the objects that need to be updated. Note: In SAS Decision Services, all IDs are strings.

Name	Type	Description
updates	array	<p>An array of update instructions, as defined in the JSON patch format.</p> <p>An update instruction includes the following fields:</p> <p>op (string) For the purposes of this media type, only updates are supported, so the value of the op field is always replaced. No other values are supported.</p> <p>path (string) A string that specifies the path using / delimited parts. For the use cases supported in the SAS Decision Services Administration API, the value of the path is the name of the field.</p> <p>value (any) The value that is to be assigned to the path. The type of the value is determined by the type of the field updated. For example, if an activation status is updated, the value of the path is active and value of value is true.</p> <p>ifUnmodifiedSince (dateTime) (Optional) This member contains the timestamp. If this value is present, the resource is updated only when the resource has not been modified since the supplied timestamp. If the resource has been modified, the change is not made, and the update is considered a failure and handled per the</p>

Name	Type	Description
failurePolicy	string	A string token that specifies how failures are handled. The allowed values are ignore and rollback . Using ignore continues to process other resources specified in the selection, even when the Update operation failed for some. Using rollback leaves the system in the original state without any updates. If an error is encountered while updating one of the resources specified in the selection field above.

Here is an example of application/vnd.sas.decision.batch.update.request:

```
{
  "version" : 1,
  "selection" : [ "id1", "id2", ..., "idn" ],
  "updates" :
    [
      { "op": "replace", "path": "/active", "value": true },
      { "op": "replace", "path": "/timeOut", "value": 10 }
    ],
  "failurePolicy" : "ignore"
}
```

application/vnd.sas.decision.batch.update.result

The application/vnd.sas.decision.batch.update.result media type represents the result of a bulk update on a set of SAS Decision Services resources.

Name	Type	Description
version	integer	The media type's schema version number.

Name	Type	Description
startTimestamp	dateTime	The server timestamp indicating when the server started to process the request.
endTimestamp	dateTime	The server timestamp indicating when the server finished processing the request.
successCount	integer	Specifies how many resources were updated successfully as a result of the request. If the failure mode is set to rollback, then this value should either be 0 (meaning no updates), or the size of the selection array.
failureCount	integer	Specifies how many resources were not updated successfully as a result of the request. If the failure mode is set to rollback, then this value should either be 0 (meaning that all updates were successful), or the size of the selection array (meaning no updates).

Name	Type	Description
results	array	<p>An array of result objects. One per resource is specified in the selection field of the request. The result includes the following fields:</p> <p>id (string) The ID of the resource that was updated or not updated.</p> <p>status (string) The value of success or failure, indicating whether the resource was updated successfully or failed to update.</p> <p>error (object) (Optional) An error object. Included if the update failed and resulted in errors that can be linked to this particular resource.</p>
error	object	(Optional) An error object. Included if the update failed.
links	collection of link objects	One or more link objects.

```
{
  "version" : 1,
  "startTimeStamp" : "time started",
  "endTimeStamp" : "end time",
  "results" : [{
    "id" : "id1",
    "status" : "success"
  }, {
    "id" : "id2",
    "status" : "failed",
    "error" : {
      "errorCode": 404,
      "httpStatusCode": 404,
      "details": [
```



```

        "Decision Definition GetRecommendation was referenced by
        Decision Flow Recommendation Campaign, but was not found."
    ],
    "links": [],
    "message": "Activation failed.",
    "remediation": "Please ensure all relevant artifacts
                    are available in the repository.",
    "version": 1
}
}]
"error" : {
    "errorCode": 409,
    "httpStatusCode": 409,
    "details": [
        "Decision Definition GetRecommendation was referenced by
        Decision Flow Recommendation Campaign, but was not found."
        "Decision Definition GetRecommendation was referenced by
        Decision Flow Recommendation Campaign as well as Recommendation
        Campaign2. Only one Decision Flow can reference a Decision Definition."
    ],
    "links": [],
    "message": "Activation failed.",
    "version": 1
}
}

```

Resources and Collections

Resource /

The / resource returns an object that contains a collection of links to the three top-level collections that are surfaced by this API:

- /decisionDefinitions
- /decisionFlows
- /variables

The / resource uses the GET / method, which requires authentication, and has a request URL of GET <http://www.example.com/SASDecisionServicesAdministration/rest/>.

The following is an example of the JSON representation:

```
{
  "links": [
    {
      "href": "http://www.example.com/SASDecisionServicesAdministration/
rest/decisionDefinitions"
      "method": "GET",
      "rel": "decisionDefinitions",
      "type": "application/vnd.sas.collection",
      "uri": "/decisionDefinitions"
    },
    {
      "href": "http://www.example.com/SASDecisionServicesAdministration/
rest/decisionFlows"
      "method": "GET",
      "rel": "decisionFlows",
      "type": "application/vnd.sas.collection",
      "uri": "/decisionFlows"
    },
    {
      "href": "http://www.example.com/SASDecisionServicesAdministration/
rest/variables"
      "method": "GET",
      "rel": "variables",
      "type": "application/vnd.sas.collection",
      "uri": "/variables"
    }
  ]
}
```

The following is an example of the JSON representation:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <links>
    <link href="http://www.example.com/SASDecisionServicesAdministration/rest/
      decisionDefinitions"
      method="GET" rel="decisionDefinitions" uri="/decisionDefinitions"
      type="application/vnd.sas.collection"/>
    <link href="http://www.example.com/SASDecisionServicesAdministration/rest/
      decisionFlows"
      method="GET" rel="decisionFlows" uri="/decisionFlows"
      type="application/vnd.sas.collection"/>
    <link href="http://www.example.com/SASDecisionServicesAdministration/rest/
      variables"
      method="GET" rel="variables" uri="/variables">
```

```

        type="application/vnd.sas.collection"/>
    </links>
</collection>

```

Here are the HTTP response codes:

200

OK

401

Unauthorized

403

Forbidden

404

Not found

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET / returns the following media type representations by setting the Accept: header of the request:

- application/json
- application/xml

Collection /decisionDefinitions

Overview

The /decisionDefinitions represents the collection of decision definitions available in a run-time deployment. A decision definition is usually associated with one or more decision flows, only one of which can be active.

The /decisionDefinitions collection has the following methods:

- GET

■ POST

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions`.

The GET method returns a collection of decision definition summaries corresponding to the decision definitions in the system. The collection supports pagination and filtering by the label (display name) of the decision definition.

The GET method has the following query parameters:

Name	Type	Description
?start	integer	The starting index of the first item in a page. The index is 0-based. The default is 0.
?limit	integer	The maximum number of definition summaries to return in this page of results. The actual number of returned definition summaries can be less if the collection has been exhausted. The default is 10.
?label	string	Filters by the label of the decision definition. The label is also known as the display name.

The following is an example of the JSON representation:

```
{  "version" : 2,
  "start" : start,
  "limit" : limit,
  "count" : count,
  "name" : "decisionDefinitionSummaries",
  "accept": "application/vnd.sas.decision.definition.summary+json"
  "items" :
    [
      { application/vnd.sas.decision.definition.summary+json contenti },
      { application/vnd.sas.decision.definition.summary+json contenti+1 },
      ...
      { application/vnd.sas.decision.definition.summary+json contentn },
    ]
}
```

```

    ],
    "links": [
        { "rel" : "first", "method" : "GET", ... },
        { "rel" : "prev", "method" : "GET", ... },
        { "rel" : "next", "method" : "GET", ... },
        { "rel" : "last", "method" : "GET", ... },
        ...
    ]
}

```

The following is an example of the JSON representation:

```

<collection name="decisionDefinitionSummaries"
  accept="application/vnd.sas.decision.definition.summary+xml"
  start="start"
  limit="limit"
  count="count" >
  <items>
    <decisionDefinitionSummary>
      <!-- application/vnd.sas.decision.definition.summary+xml contenti -->
    </decisionDefinitionSummary>
    <decisionDefinitionSummary>
      <!-- application/vnd.sas.decision.definition.summary+xml contenti+1 -->
    </decisionDefinitionSummary>
    ...
    <decisionDefinitionSummary>
      <!-- application/vnd.sas.decision.definition.summary+xml contentn -->
    </decisionDefinitionSummary>
  </items>
  <links>
    <link rel="first" method="GET" ... />
    <link rel="prev" method="GET" ... />
    <link rel="next" method="GET" ... />
    <link rel="last" method="GET" ... />
  </links>
</collection>

```

Here are the HTTP response codes:

200

OK

401

Unauthorized

403

Forbidden

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.collection+json;version=2
- application/vnd.sas.collection+xml;version=2
- application/vnd.sas.collection+json
- application/vnd.sas.collection+xml
- application/json
- application/xml

The POST method sets a timeout for a group of decision definitions. Changing the artifacts in use by the engine is expensive and requires validating any new changes against the existing configuration and artifacts already loaded. Bulk update methods like these collect all the changes required and perform the validation only once instead of once per change. This allows for meeting strict performance requirements.

The POST method requires authentication and has a request URL of POST <http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions>.

The POST method accepts the following content types, as named by the Content-Type: header:

- application/vnd.sas.decision.batch.update.request+json
- application/vnd.sas.decision.batch.update.request+xml
- application/json
- application/xml

Because only the timeout field can be updated, the updates array can contain only a single entry with the value of the op field set to replace, the value of the path field set to

timeout, and the value field set to a positive floating point number representing the timeout value in seconds.

A response returns the results of the update as a application/vnd.sas.decision.batch.update.result.

Here are the HTTP response codes:

200

OK

400

Bad Request

401

Unauthorized

403

Forbidden

404

Not found.

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

The POST method can return the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.decision.batch.update.results+json
- application/vnd.sas.decision.batch.update.results+xml
- application/json
- application/xml

Resource /decisionDefinitions/{decisionId}

The /decisionDefinitions/{decisionId} resource represents a single decision definition in the system.

The GET method has the following query parameters:

Name	Type	Description
{decisionId}	string	The URL-encoded name of the decision definition.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions/{decisionId}` .

The GET method returns a representation of a single decision definition summary as identified by the decisionId.

Here are the HTTP response codes:

- 200
OK
- 401
Unauthorized
- 403
Forbidden
- 404
Not found
- 500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.decision.definition.summary+json
- application/vnd.sas.decision.definition.summary+xml
- application/json
- application/xml

Resource **/decisionDefinitions/{decisionId}/timeout**

The `/decisionDefinitions/{decisionId}/timeout` resource represents the value of the timeout, in seconds, for this decision definition. This value is used to determine whether a decision flow execution has gone on for too long and should be stopped. The timeout value is described in the `application/vnd.sas.decision.definition.summary` media type. The value is a positive floating point number and is returned or accepted as text/plain.

Name	Type	Description
{decisionId}	string	The URL-encoded name of the decision definition.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions/{decisionId}/timeout`.

The GET method returns the value of the timeout, in seconds, for the decision definition identified by the `decisionId`.

Here are the HTTP response codes:

- 200
OK
- 401
Unauthorized
- 403
Forbidden
- 404
Not found

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representation by setting the Accept: header of the request:

- text/plain

The PUT method sets the value of the timeout, in seconds, for the decision definition identified by the decisionId.

Note: For updating the timeout value for multiple decision definitions, the batch update process is the most efficient option.

The PUT method requires authentication and has a request URL of PUT `http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions/{decisionId}/timeout`.

The PUT method accepts the following content type, as named by the Content-Type: header:

- text/plain

Here are the HTTP response codes:

204

No content

400

Bad request

401

Unauthorized

403

Forbidden

404

Not found

500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

Resource /decisionDefinitions/{decisionId}/timeoutEnabled

The /decisionDefinitions/{decisionId}/timeoutEnabled resource represents a Boolean value to indicate whether timeout processing is turned on for this decision definition, as identified by the decisionId.

Name	Type	Description
{decisionId}	string	The URL-encoded name of the decision definition.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions/{decisionId}/timeoutEnabled` .

The GET method returns the value of the timeoutEnabled flag for this decision definition. The value is returned as a Boolean value.

Here are the HTTP response codes:

- 200
OK
- 401
Unauthorized
- 403
Forbidden
- 404
Not found
- 500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representation by setting the Accept: header of the request:

- text/plain

The PUT method updates the value of the timeoutEnabled flag for this decision definition. Accepted values are Boolean values.

The PUT method requires authentication and has a request URL of PUT `http://www.example.com/SASDecisionServicesAdministration/rest/decisionDefinitions/{decisionId}/timeoutEnabled`.

The PUT method accepts the following content type, as named by the Content-Type: header:

- text/plain

Here are the HTTP response codes:

204

No content

400

Bad request

401

Unauthorized

403

Forbidden

404

Not found

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

Collection /decisionFlows

Overview

The /decisionFlows collection represents the collection of decision flows in the system. Every decision flow is associated with a decision definition that it references using the decisionId. It is possible that the referenced decision definition is not available in the system. In such cases, the decision flow is considered invalid. Decision flows can be active or inactive. Only valid decision flows can be active. Also, multiple decision flows can be associated with a decision definition. Only a single such flow can be active.

The /decisionFlows collection has the following methods:

- GET
- POST

The GET method requires authentication and has a request URL of GET <http://www.example.com/SASDecisionServicesAdministration/rest/decisionFlows>.

The GET method returns a collection of decision definition summaries corresponding to the decision definitions in the system. The collection supports pagination and filtering by the following attributes:

label (string)

The display name of the decision flow.

active (Boolean)

The flag indicating whether the decision flow is active or not.

decisionId (string)

The ID of the decision definition that this flow is associated with.

The GET method has the following query parameters:

Name	Type	Description
?start	integer	The starting index of the first decision flow in a page. The index is 0-based. The default is 0.
?limit	integer	The maximum number of decision flows to return in this page of results. The actual number of returned decision flows can be less, if the collection has been exhausted. The default is 10.
?label	string	Returns decision flows that have this display name.
?active	Boolean	Returns decision flows that are active or not based on the value of this attribute.
?decisionId	string	Returns decision flows that are associated with a decision definition with this ID. There could be more than one such decision flow.

The following is an example of the JSON representation:

```
{
  "version" : 2,
  "start" : start,
  "limit" : limit,
  "count" : count,
  "name" : "decisionFlowSummaries",
  "accept": "application/vnd.sas.decision.flow.summary+json"
  "items" :
    [
      { application/vnd.sas.decision.flow.summary+json contenti },
      { application/vnd.sas.decision.flow.summary+json contenti+1 },
      ...
      { application/vnd.sas.decision.flow.summary+json contentn },
    ],
  "links": [
    { "rel" : "first", "method" : "GET", ... },
```

```

        { "rel" : "prev", "method" : "GET", ... },
        { "rel" : "next", "method" : "GET", ... },
        { "rel" : "last", "method" : "GET", ... },
        ...
    ]
}

```

The following is an example of the JSON representation:

```

<collection name="decisionFlowSummaries"
  accept="application/vnd.sas.decision.flow.summary+xml"
  start="start"
  limit="limit"
  count="count" >
  <items>
    <decisionFlowSummary>
      <!-- application/vnd.sas.decision.flow.summary+xml contenti -->
    </decisionFlowSummary>
    <decisionFlowSummary>
      <!-- application/vnd.sas.decision.flow.summary+xml contenti+1 -->
    </decisionFlowSummary>
    ...
    <decisionFlowSummary>
      <!-- application/vnd.sas.decision.flow.summary+xml contentn -->
    </decisionFlowSummary>
  </items>
  <links>
    <link rel="first" method="GET" ... />
    <link rel="prev" method="GET" ... />
    <link rel="next" method="GET" ... />
    <link rel="last" method="GET" ... />
  </links>
</collection>

```

Here are the HTTP response codes:

200

OK

401

Unauthorized

403

Forbidden

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.collection+json;version=2
- application/vnd.sas.collection+xml;version=2
- application/vnd.sas.collection+json
- application/vnd.sas.collection+xml
- application/json
- application/xml

The POST method activates or deactivates multiple flows in a batch. Changing artifacts in use by the engine is expensive and requires validating any new changes against the existing configuration and artifacts already loaded. Bulk update methods like these collect all the changes required and perform the validation only once instead of once per change. This allows for meeting strict performance requirements.

The POST method requires authentication and has a request URL of POST <http://www.example.com/SASDecisionServicesAdministration/rest/decisionFlows>.

The POST method accepts the following content types, as named by the Content-Type: header:

- application/vnd.sas.decision.batch.update.request+json
- application/vnd.sas.decision.batch.update.request+xml
- application/json
- application/xml

Since only the active field can be updated, the updates array can contain only a single entry with the value of the op field set to replace, the value of the path field set to active,

and the value field set to a Boolean value representing whether the flows are active or not.

Here are the HTTP response codes:

200

OK

400

Bad Request

401

Unauthorized

403

Forbidden

404

Not found.

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

The POST method can return the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.decision.batch.update.results+json
- application/vnd.sas.decision.batch.update.results+xml
- application/json
- application/xml

Resource /decisionFlows/{decisionFlowId}

The /decisionFlows/{decisionFlowId} resource represents a single decision flow in the system as identified by the decisionFlowId.

The GET method has the following query parameters:

Name	Type	Description
{decisionFlowId}	string	The URL-encoded name of the decision flow.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/decisionFlows/{decisionFlowId}`.

The GET method returns a single decision flow as identified by `decisionFlowId`.

Here are the HTTP response codes:

- 200
OK
- 401
Unauthorized
- 403
Forbidden
- 404
Not found
- 500
Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the `Accept:` header of the request:

- `application/vnd.sas.decision.flow.summary+json`
- `application/vnd.sas.decision.flow.summary+xml`
- `application/json`
- `application/xml`

Resource /decisionFlows/{decisionFlowId}/active

The /decisionFlows/{decisionFlowId}/active resource represents the activation status of the decision flow identified by decisionFlowId.

The GET method has the following query parameters:

Name	Type	Description
{decisionFlowId}	string	The URL-encoded name of the decision flow.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/decisionFlows/{decisionFlowId}/active`.

The GET method returns the activation status of this decision flow as a Boolean value. True if the decision flow is active, false otherwise.

Here are the HTTP response codes:

200	OK
401	Unauthorized
403	Forbidden
404	Not found
500	Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

■ text/plain

The PUT method updates the value of the activation status of this decision flow. Accepted values are Boolean values.

Note: For updating the activation status for multiple decision flows, the batch update process is the most efficient option.

The PUT method requires authentication and has a request URL of PUT `http://www.example.com/SASDecisionServicesAdministration/rest/decisionFlows/{decisionFlowId}/active`.

The PUT method accepts the following content type, as named by the Content-Type header:

■ text/plain

Here are the HTTP response codes:

204

No content

400

Bad request

401

Unauthorized

403

Forbidden

404

Not found

500

Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

Collection /variables

Overview

The /variables collection represents all of the variables in the system.

The /variables collection has a GET method. The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/variables`.

It returns a collection of resources that represents the variables in the system. The collection elements are instances of the `application/vnd.sas.decision.variable` media type.

The GET method has the following query parameters:

Name	Type	Description
?start	integer	The starting index of the first variable in a page. The index is 0-based. The default is 0.
?limit	integer	The maximum number of variables to return in this page of results. The actual number of returned decision flows can be less, if the collection has been exhausted. The default is 10.
?label	string	Returns all variables that have this display name.

The following is an example of the JSON representation:

```
{
  "version" : 2,
  "start" : start,
  "limit" : limit,
  "count" : count,
  "name" : "variables",
  "accept": "application/vnd.sas.decision.variable+json"
  "items" :
```

```

    [
      { application/vnd.sas.decision.variable+json contenti },
      { application/vnd.sas.decision.variable+json contenti+1 },
      ...
      { application/vnd.sas.decision.variable+json contentn },
    ],
    "links": [
      { "rel" : "first", "method" : "GET", ... },
      { "rel" : "prev", "method" : "GET", ... },
      { "rel" : "next", "method" : "GET", ... },
      { "rel" : "last", "method" : "GET", ... },
      ...
    ]
  }
}

```

The following is an example of the JSON representation:

```

<collection name="variables"
  accept="application/vnd.sas.decision.variable+xml"
  start="start"
  limit="limit"
  count="count" >
  <items>
    <variable>
      <!-- application/vnd.sas.decision.variable+xml contenti -->
    </variable>
    <variable>
      <!-- application/vnd.sas.decision.variable+xml contenti+1 -->
    </variable>
    ...
    <variable>
      <!-- application/vnd.sas.decision.variable+xml contentn -->
    </variable>
  </items>
  <links>
    <link rel="first" method="GET" ... />
    <link rel="prev" method="GET" ... />
    <link rel="next" method="GET" ... />
    <link rel="last" method="GET" ... />
  </links>
</collection>

```

Here are the HTTP response codes:

200

OK

401	Unauthorized
403	Forbidden
404	Not found
500	Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.collection+json;version=2
- application/vnd.sas.collection+xml;version=2
- application/vnd.sas.collection+json
- application/vnd.sas.collection+xml
- application/json
- application/xml

Resource /variables/{variableId}

The /variables/{variableId} resource represents a single variable in the system.

The GET method has the following query parameters:

Name	Type	Description
{variableId}	string	The URL-encoded name of the variable.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/variables/{variableId}`.

The GET method returns a representation of a single variable as identified by the variableId.

Here are the HTTP response codes:

200	OK
401	Unauthorized
403	Forbidden
404	Not found
500	Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

- application/vnd.sas.decision.variable+json
- application/vnd.sas.decision.variable+xml
- application/json
- application/xml

Resource /variables/{variableId}/value

The /variables/{variableId}/value resource represents the value of this variable identified by the variableId. Primitive values like string or Boolean are returned as text/plain. Complex values like array or table are returned as application/json or application/xml. Depending on the type of the variable, the allowed values can be parsed or set.

The GET method has the following query parameters:

Name	Type	Description
{variableId}	string	The URL-encoded name of the variable.

The GET method requires authentication and has a request URL of GET `http://www.example.com/SASDecisionServicesAdministration/rest/variables/{variableId}/value`.

The GET method returns the value of this variable.

Here are the HTTP response codes:

200	OK
401	Unauthorized
403	Forbidden
404	Not found
500	Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

GET returns the following media type representations by setting the Accept: header of the request:

- text/plain
- application/json
- application/xml

The PUT method sets the value of this variable with the supplied value.

The PUT method requires authentication and has a request URL of PUT `http://www.example.com/SASDecisionServicesAdministration/rest/variables/{variableId}/value`.

The PUT method accepts the following content type, as named by the Content-Type header:

- text/plain
- application/json
- application/xml

Here are the HTTP response codes:

204	No content
400	Bad request
401	Unauthorized
403	Forbidden
404	Not found
500	Server error

Note: These are the most common HTTP response codes. You should be prepared to handle all valid HTTP response codes, including 3xx redirection response codes.

Recommended Reading

Here is the recommended reading list for this title:

- *SAS Federation Server Administrator's Guide*
- *SAS DS2 Language Reference*
- *SAS BI Web Services Developer's Guide*
- *SAS Intelligence Platform: Middle-Tier Administration Guide*

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books

SAS Campus Drive

Cary, NC 27513-2414

Phone: 1-800-727-0025

Fax: 1-919-677-4444

Email: sasbook@sas.com

Web address: sas.com/store/books

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.

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. Short form: DBMS.

DBMS

See database management system

federated DSN

a data source name that references multiple data sources. The data sources can be on the same DBMS, or on a different one.

grouping data source name

See federated DSN

grouping DSN

See federated DSN

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 provides metadata management services to one or more client applications. A SAS Metadata Server is an example.

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).

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.

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 Management Console

a Java application that provides a single user interface for performing SAS administrative tasks.

SAS Metadata Repository

a container for metadata that is managed by the SAS Metadata Server.

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

Index

Special Characters

\$User_Log_JDBCConnectionResource 202

A

activating decision flows 41
 activities 22, 84
 See also General I/O activities
 See also SAS activities
 See also web service activities
 date and time formats 117
 guidelines for creating 117
 migration 164
 promotion rules 35
 administration tasks 1
 aliases, defining for schemas 78
 Apache Commons Database Connection Pooling (DBCP) 177
 application servers 13
 architecture 12
 arrays, methods for 89

B

BI web services 106, 139
 Boolean values in SAS activities 117

C

capabilities for roles 27
 clustering best practices 207
 concurrent updates 118
 ConcurrentWait node 26
 configuration 13, 202
 \$User_Log_JDBCConnectionResource 202
 additional databases 203
 additional Federation Servers for server cluster 205
 for use of data sets 186, 212
 GeneralIO_Activity_Resource 202
 SAS_Activity_Resource 202
 connections, dropping idle 182
 Content Repositories folder 3
 cross-sell offers
 example of decision flow 16

D

- data sets
 - configuration for use 186, 212
- DATA step code migration 163, 167
- data type mappings
 - DS2 packages 88
 - web service activities 105
- Database Connection Pooling (DBCP) 177
- database I/O 148
- database servers 13
 - manual migration 169
- databases
 - installing and configuring
 - additional 204
 - requirements 217
- date and time formats 117
- DBCP (Database Connection Pooling) 177
- decision flows 22
 - activating 41
 - example 16
 - life cycle 18
 - managing global variables 45
 - promoting 34
 - rules for promoting 34
 - sub-flows 24
- Decision Services Manager
 - plug-in 2
 - creating and deleting
 - repositories 60
- Decision Services:
 - Administration role 27

- Decision Services: Advanced
 - role 27
- deployment 140
 - best practices for performance 142
 - connection and statement
 - pool tuning 148
 - database I/O considerations 148
 - easy button deployments 143
 - hardware capacity planning 145
 - production deployments 144
 - scenarios 146
 - web applications 149
- deployment topology 12
- development environment
 - components 14
 - installation 138
- DS2 packages
 - creating 86
 - data type mappings 88
 - sas_activity_tests (sample)
 - package 97
 - tap_array package 89
 - tap_hash package 89
- DS2 procedure 86

E

- error compensation 24
- events 20
 - promotion rules 35
 - time-out setting 47

F

failover, hardware [9](#)
 fault response [24](#)
 Federation Servers
 See [SAS Federation Servers](#)
 flows
 See [decision flows](#)
 folders
 Content Repositories [3](#)
 SAS Decision Services
 servers [2](#)

G

General I/O activities [109](#)
 Insert operation [111](#)
 Read operation [110](#)
 Update operation [111](#)
 using library resources [116](#)
 GeneralIO_Activity_Resource
 [202](#)
 global variables [24](#)
 managing [45](#)

H

hardware capacity planning
 [145](#)
 hardware failover [9](#)
 hash objects [89](#)
 HTTP servers [13](#)

I

Insert operation (General I/O
 activities) [111](#)
 installation [138](#)
 best practices for performance
 [142](#)
 BI web services [139](#)
 choosing environments [138](#)
 connection and statement
 pool tuning [148](#)
 database I/O considerations
 [148](#)
 deploying web applications
 [149](#)
 deployment scenarios [140](#),
 [146](#)
 easy button deployments [143](#)
 hardware capacity planning
 [145](#)
 production deployments [144](#)
 reconfiguring engine and
 design servers [155](#)
 SAS Authentication Server
 [139](#)
 SAS Federation Server [140](#)
 third-party components [139](#)
 web application servers [139](#)
 integration
 with SAS Model Manager [134](#)
 with web services [119](#)

J

Java 2 Enterprise Edition
 (J2EE) application servers
 9, 15

JBoss Application Server
 log file location 220

JDBC Connection system
 resources 67

 creating 72

 for General I/O activities 109

 migration 165, 169

 pool tuning for performance
 176

L

library resources 23, 78

 defining schema aliases 78

 using with General I/O
 activities 116

locking errors 118

logging

 file locations for
 troubleshooting 219

 performance considerations
 176

M

methods

 arrays 89

 validation 97

Migrate utility 164

migration

 activities 164

 comparisons to earlier
 releases 158

 from SAS Real-Time Decision
 Manager 5.4 159

 host changes 169

 JDBC Connection system
 resources 165, 169

 Migrate utility 164

 multiple DATA step code 167

 single DATA step code 163

 UpdateResource utility 165

 utilities 164

 Web Service Connection
 system resources 165

 web service resources 170

models 134

N

nodes

 concurrent execution 25

 ConcurrentWait 26

P

performance

 See [system performance](#)

process variables 23

 web service activities 105

production environment

- components 15
- installation 138
- servers 13
- promotion
 - activities 35
 - decision flows 34
 - events 35
 - example in SAS Management Console 35

R

- Read operation (General I/O activities) 110
- reconfiguration of servers 155
- repositories 60
 - creating 60
 - deleting 64
 - viewing 4
- roles 27

S

- SAS activities 85
 - See *also* [DS2 packages](#)
 - Boolean values in 117
 - creating 85
 - creating XML 87
- SAS Authentication Servers 15, 139
- SAS Customer Intelligence Studio
 - creation of global variables 45

- SAS data sets
 - configuration for use 186, 212
- SAS Decision Services
 - administrative tasks 1
 - components 12
 - concepts 9
 - configuration 13, 202
 - Decision Services Manager
 - plug-in 2
 - installation 138
 - integration with SAS Model Manager 134
 - migration 158
 - overview 1
 - troubleshooting 219
 - viewing repositories 4
 - web service integration 119
- SAS Decision Services design
 - server 15
 - log file location 220
- SAS Decision Services engine
 - 9
 - server log file location 220
 - servers 13, 15
- SAS Decision Services servers
 - folder 2
- SAS Federation Servers 13, 15, 140
 - allocating computing resources 67
 - configuring to form a server cluster 205
 - log file location 220
 - manual migration 169
 - options for performance 176

- requirements for databases 217
- tuning considerations 181
- SAS Management Console 15
 - Decision Services Manager plug-in 2
 - promotion example 35
- SAS Metadata Repository 15
- SAS Metadata Servers 13
 - log file location 220
- SAS Model Manager
 - integration with SAS Decision Services 134
- SAS Real-Time Decision Manager 5.4
 - migration from 159
- SAS Web Application Server 15
- SAS_Activity_Resource 202
- sas_activity_tests (sample) DS2 package 97
- scalability 9
- schema aliases, defining 78
- servers
 - application servers 13
 - clustering 9, 207
 - database servers 13
 - HTTP servers 13
 - Java 2 Enterprise Edition (J2EE) application servers 9, 15
 - production environment 13
 - reconfiguring engine and design servers 155
 - SAS Authentication Servers 15
 - SAS Decision Services design server 15
 - SAS Decision Services engine servers 13, 15
 - SAS Federation Servers 13, 15
 - SAS Metadata Servers 13
- Simple Object Access Protocol (SOAP) 119
- SOAP (Simple Object Access Protocol) 119
- sub-flows 24
- system performance
 - allocating computing resources 67
 - best practices 142, 176
 - connection and statement pool tuning 148, 182
 - database I/O considerations 148
 - JDBC performance tuning 176
 - SAS Federation Server tuning 181
 - server options 176
- system resources 23, 66
 - JDBC Connection 67, 109
 - promotion rules 34
 - Web Service Connection 76, 106

T

tap_array DS2 package 89
 tap_hash DS2 package 89
 test environment
 components 15
 installation 138
 third-party components
 installation 139
 time-out values, setting 48
 topology, deployment 12
 troubleshooting 219
 tuning connection and
 statement pools 148, 182

U

Update operation (General I/O
 activities) 111
 UpdateResource utility 165
 utilities for migration 164

V

validation
 methods for 97
 variables, global
 See [global variables](#)
 variables, process
 See [process variables](#)

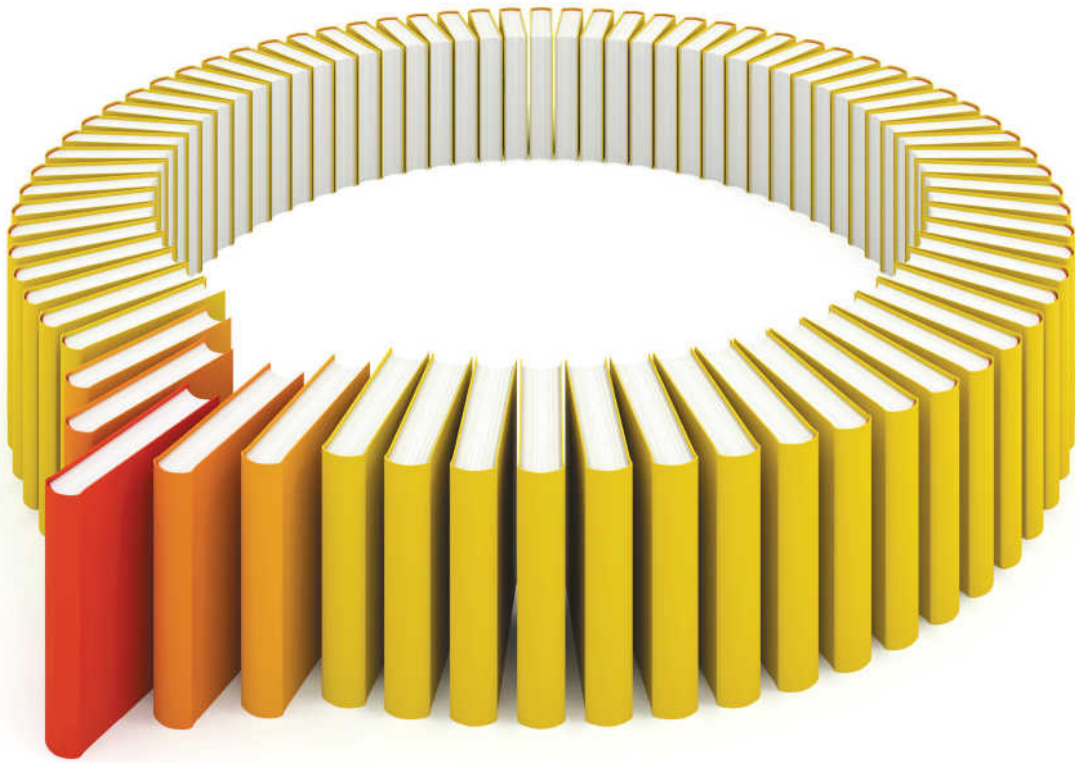
W

web application servers
 installation 139
 web applications
 deploying 149
 web service activities 105
 BI web services 106
 Web Service Connection
 system resources for 106
 Web Service Connection
 system resources
 creating 76
 for web service activities 106
 migration 165
 Web Service Definition
 Language (WSDL) files 120
 changing URL for migration
 170
 web services
 integration with SAS Decision
 Services 119
 request example 122
 WebSphere Application Server
 sequence for starting web
 applications 150
 WSDL (Web Service Definition
 Language) files 120
 changing URL for migration
 170

X

XML

creating for SAS activities [87](#)



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

