THE POWER TO KNOW®

SAS® Decision Services 6.2 Administrator’s Guide

SAS® Documentation

Contents

What’s New in SAS Decision Services 6.2

Overview

SAS Decision Services has been designed to support the initial release of SAS Decision Manager. SAS Decision Services 6.2 has the following enhancements:

- Flow DS2 Code Generation for Batch Execution
- Decision Logic Smart Object

Flow DS2 Code Generation for Batch Execution

This new feature builds on existing flow DS2 code generation by adding a batch processing code wrapper. This wrapper iterates over a table of transaction data and calls the generated flow DS2 once per transaction record. This feature is used by the Decision Logic smart object.

Decision Logic Smart Object

A Decision Logic smart object is a SAS metadata object that contains source code and descriptive metadata that represents a SAS Decision Services working set. A working set consists of a top-level flow and any sub-flow, activities, and related components. A new method in the design server API enables client applications to create Decision Logic smart objects and store them in SAS Metadata Server. The Decision Logic smart object supports SAS Data Integration Studio. After the smart object has been created, SAS Data Integration Studio can discover a Decision Logic smart object, embed it in a SAS Data Integration Studio job, connect data sources to it, and execute it.

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 books, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Book Sales Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Chapter 1

Overview of SAS Decision Services

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 makes continuous operation possible 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 used from any 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 to author and unit test decision flows. The environments can also be used to run batch simulations.

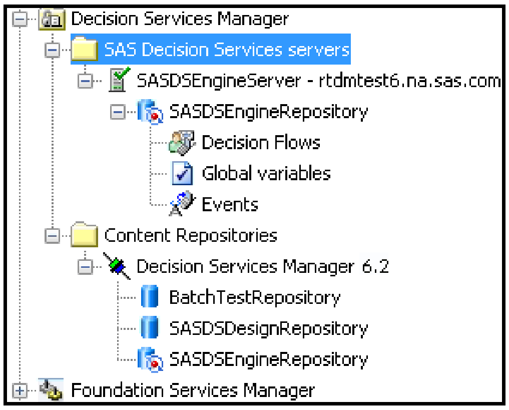
  A test or production environment consists of a cluster of one or more engine servers, a cluster of one or more SAS Federation Servers, a SAS Workspace Server, and a content repository. Test environments are typically used for final testing of a decision flow before putting it into production. Production 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.

Display 1.1 *Decision Services Manager Plug-in Folders*



- Decision Services Manager
  - SAS Decision Services servers
    - SASDSEngineServer - rtdmtest6.na.sas.com
      - SASDSEngineRepository
        - Decision Flows
        - Global variables
        - Events
  - Content Repositories
    - Decision Services Manager 6.2
      - BatchTestRepository
      - SASDSDesignRepository
      - SASDSEngineRepository
- Foundation Services Manager

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.

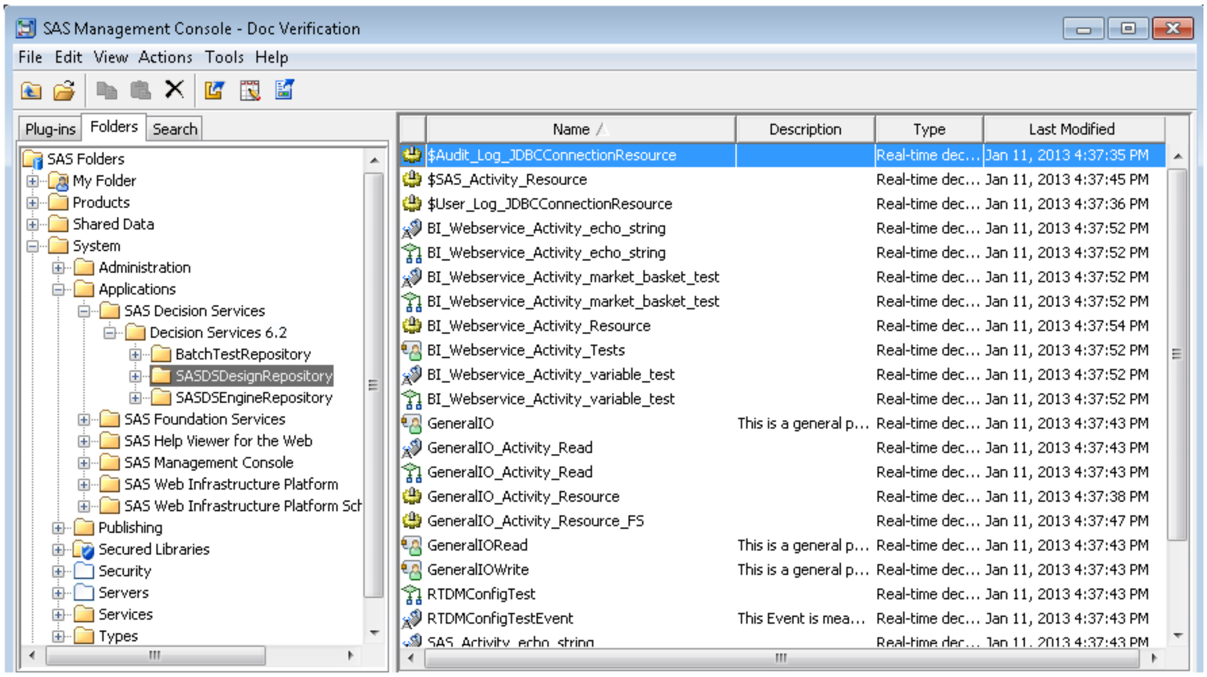
Indicates a test repository.

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.

Display 1.2 *Decision Services Manager Folders View*

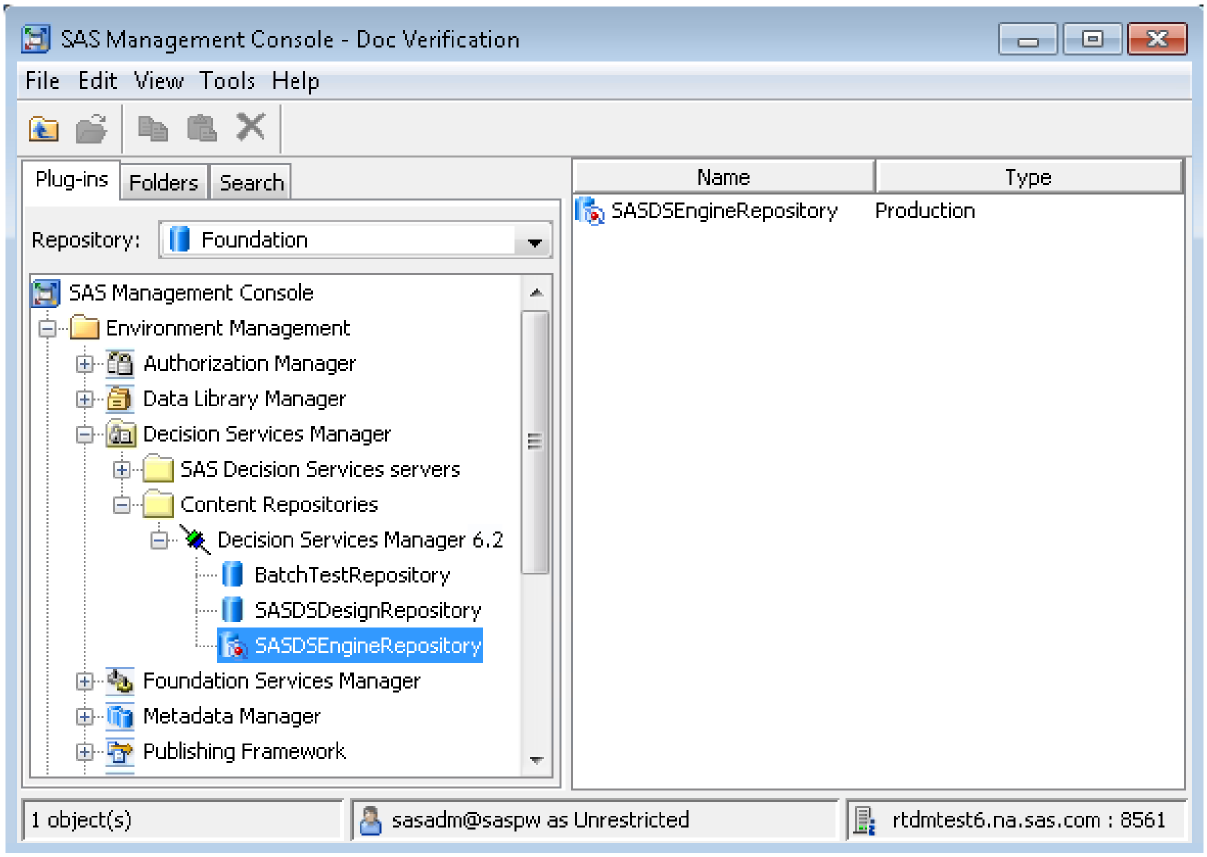

SAS Management Console - Doc Verification
File Edit View Actions Tools Help
Plug-ins
Folders
Search

- SAS Folders
  - My Folder
  - Products
  - Shared Data
  - System
    - Administration
    - Applications
      - SAS Decision Services
        - Decision Services 6.2
          - BatchTestRepository
          - SASDSDesignRepository
          - SASDSEngineRepository
      - SAS Foundation Services
      - SAS Help Viewer for the Web
      - SAS Management Console
      - SAS Web Infrastructure Platform
      - SAS Web Infrastructure Platform Sch
    - Publishing
    - Secured Libraries
    - Security
    - Servers
    - Services
    - Types

Name	Description	Type	Last Modified
$Audit_Log_JDBCConnectionResource		Real-time dec...	Jan 11, 2013 4:37:35 PM
$SAS_Activity_Resource		Real-time dec...	Jan 11, 2013 4:37:45 PM
$User_Log_JDBCConnectionResource		Real-time dec...	Jan 11, 2013 4:37:36 PM
BI_Webservice_Activity_echo_string		Real-time dec...	Jan 11, 2013 4:37:52 PM
BI_Webservice_Activity_echo_string		Real-time dec...	Jan 11, 2013 4:37:52 PM
BI_Webservice_Activity_market_basket_test		Real-time dec...	Jan 11, 2013 4:37:52 PM
BI_Webservice_Activity_market_basket_test		Real-time dec...	Jan 11, 2013 4:37:52 PM
BI_Webservice_Activity_Resource		Real-time dec...	Jan 11, 2013 4:37:54 PM
BI_Webservice_Activity_Tests		Real-time dec...	Jan 11, 2013 4:37:52 PM
BI_Webservice_Activity_variable_test		Real-time dec...	Jan 11, 2013 4:37:52 PM
BI_Webservice_Activity_variable_test		Real-time dec...	Jan 11, 2013 4:37:52 PM
GeneralIO	This is a general p...	Real-time dec...	Jan 11, 2013 4:37:43 PM
GeneralIO_Activity_Read		Real-time dec...	Jan 11, 2013 4:37:43 PM
GeneralIO_Activity_Read		Real-time dec...	Jan 11, 2013 4:37:43 PM
GeneralIO_Activity_Resource		Real-time dec...	Jan 11, 2013 4:37:38 PM
GeneralIO_Activity_Resource_FS		Real-time dec...	Jan 11, 2013 4:37:47 PM
GeneralIORead	This is a general p...	Real-time dec...	Jan 11, 2013 4:37:43 PM
GeneralIOWrite	This is a general p...	Real-time dec...	Jan 11, 2013 4:37:43 PM
RTDMConfigTest		Real-time dec...	Jan 11, 2013 4:37:43 PM
RTDMConfigTestEvent	This Event is mea...	Real-time dec...	Jan 11, 2013 4:37:43 PM
SAS_Activity_echo_string		Real-time dec...	Jan 11, 2013 4:37:43 PM

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.

Display 1.3 *Decision Services Manager Plug-ins View*

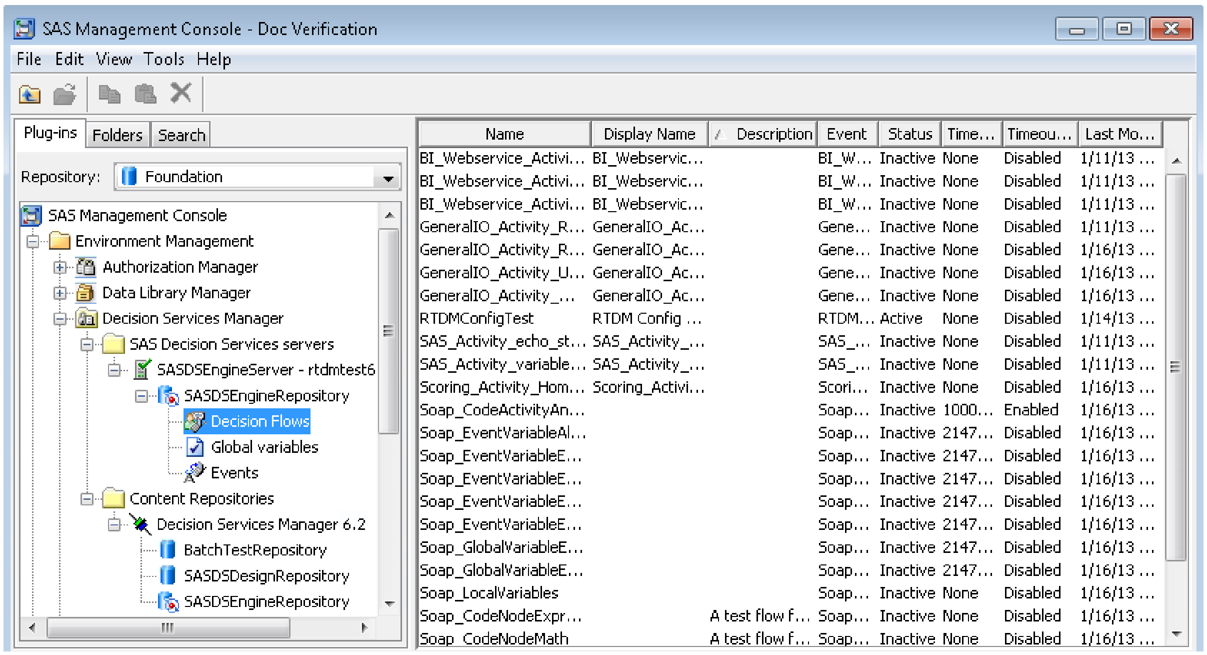


Name	Type
SASDSEngineRepository	Production

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.

Display 1.4 *Decision Flows*



SAS Management Console - Doc Verification

Repository: Foundation

- SAS Management Console
  - Environment Management
    - Authorization Manager
    - Data Library Manager
    - Decision Services Manager
      - SAS Decision Services servers
        - SASDSEngineServer - rtdmtest6
          - SASDSEngineRepository
            - Decision Flows
            - Global variables
            - Events
      - Content Repositories
        - Decision Services Manager 6.2
          - BatchTestRepository
          - SASDSDesignRepository
          - SASDSEngineRepository

Name	Display Name	Description	Event	Status	Time...	Timeou...	Last Mo...
BI_Webservice_Activi...	BI_Webservic...		BI_W...	Inactive	None	Disabled	1/11/13 ...
BI_Webservice_Activi...	BI_Webservic...		BI_W...	Inactive	None	Disabled	1/11/13 ...
BI_Webservice_Activi...	BI_Webservic...		BI_W...	Inactive	None	Disabled	1/11/13 ...
GeneralIO_Activity_R...	GeneralIO_Ac...		Gene...	Inactive	None	Disabled	1/11/13 ...
GeneralIO_Activity_R...	GeneralIO_Ac...		Gene...	Inactive	None	Disabled	1/16/13 ...
GeneralIO_Activity_U...	GeneralIO_Ac...		Gene...	Inactive	None	Disabled	1/16/13 ...
GeneralIO_Activity_...	GeneralIO_Ac...		Gene...	Inactive	None	Disabled	1/16/13 ...
RTDMConfigTest	RTDM Config ...		RTDM...	Active	None	Disabled	1/14/13 ...
SAS_Activity_echo_st...	SAS_Activity_...		SAS_...	Inactive	None	Disabled	1/11/13 ...
SAS_Activity_variable...	SAS_Activity_...		SAS_...	Inactive	None	Disabled	1/11/13 ...
Scoring_Activity_Hom...	Scoring_Activi...		Scori...	Inactive	None	Disabled	1/16/13 ...
Soap_CodeActivityAn...			Soap...	Inactive	1000...	Enabled	1/16/13 ...
Soap_EventVariableAl...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_EventVariableE...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_EventVariableE...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_EventVariableE...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_EventVariableE...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_GlobalVariableE...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_GlobalVariableE...			Soap...	Inactive	2147...	Disabled	1/16/13 ...
Soap_LocalVariables			Soap...	Inactive	None	Disabled	1/16/13 ...
Soap_CodeNodeExpr...		A test flow f...	Soap...	Inactive	None	Disabled	1/16/13 ...
Soap_CodeNodeMath		A test flow f...	Soap...	Inactive	None	Disabled	1/16/13 ...

Chapter 2
SAS Decision Services Concepts

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 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.

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 123.

Code Generation

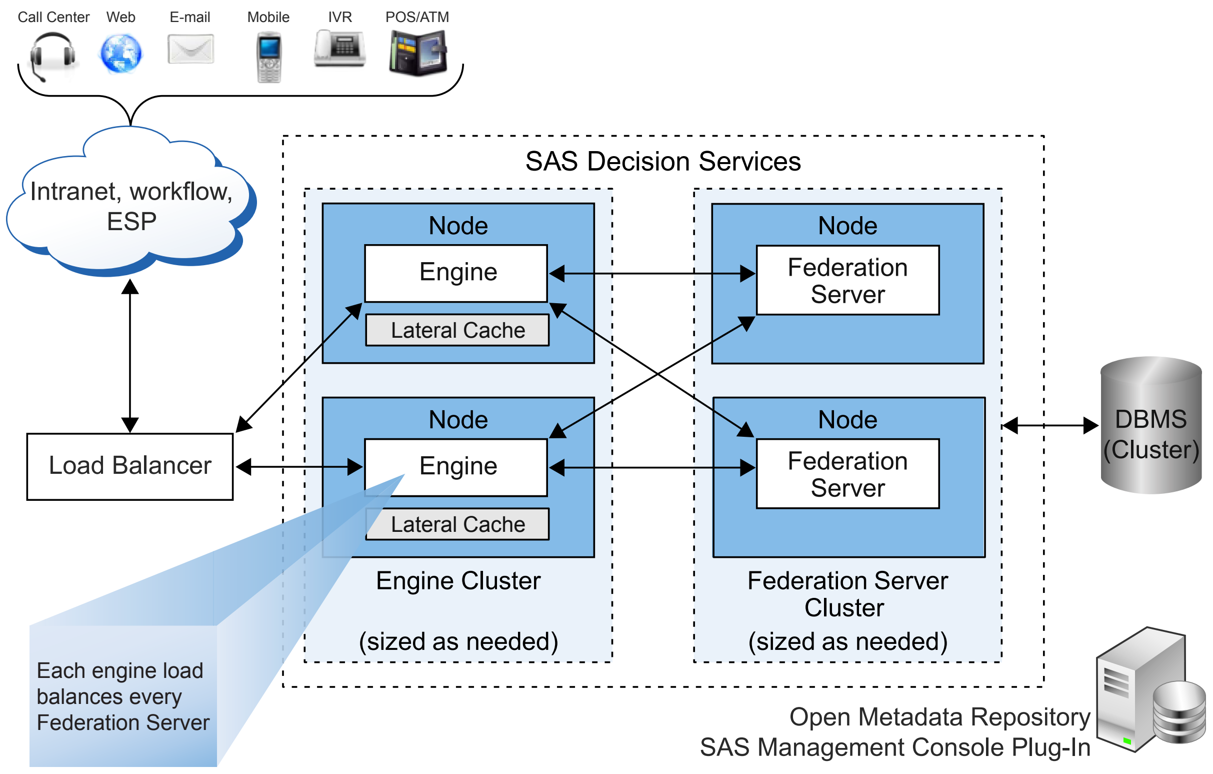
A code generation interface added to the design server allows client applications to request generation of DS2 code corresponding to a given flow and its dependencies.

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. 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*

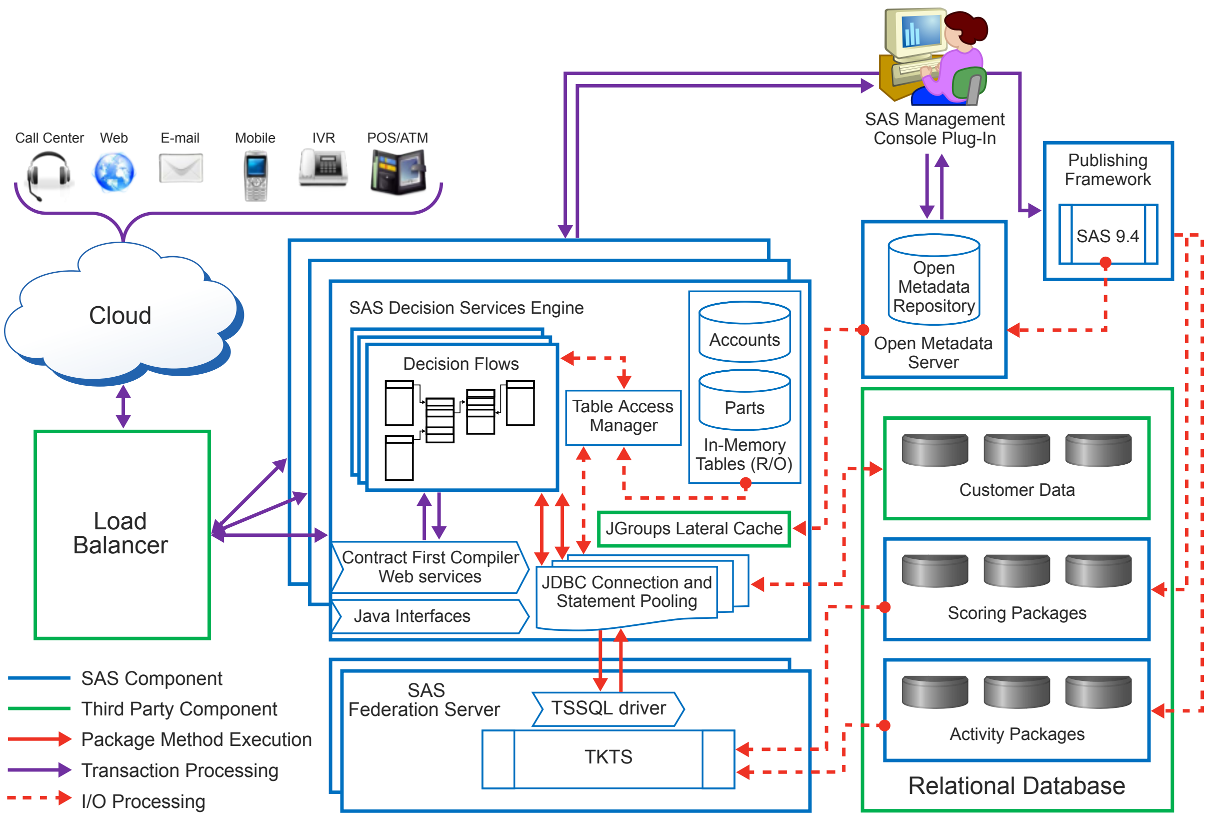


Deployment Topology

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 lightweight metadata objects that are pointers to decision flows in the content repository.
- **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.

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.

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. Each repository 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 DataFlux 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 that have operational channels.

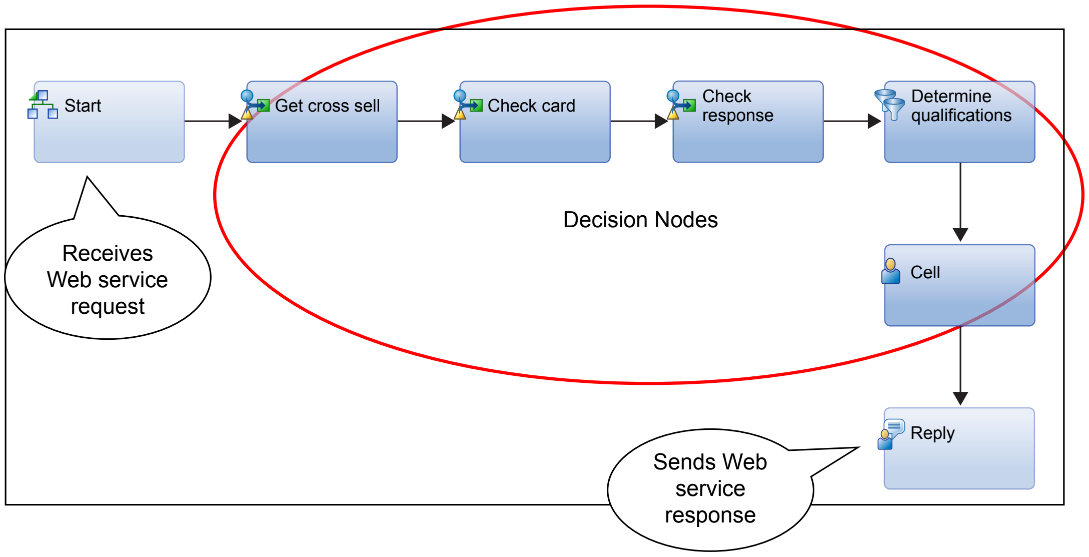
A test or production environment consists of the following components:

- 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 21.

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".)

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.

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 events that are appropriate to your processing needs. 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 39.) 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 make those requests.

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.

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 event response header contains the following data items:

CorrelationId
: This is a string field containing the value of the **Identity** field of the event.

StartTime
: This is a time stamp that shows when the message was received by the engine.

CompletionTime
: This is a time stamp 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 containing 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 used by decision flow designers.

SAS Decision Services stores DS2 source code in the activity metadata, using new 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 perform one or both of the following two functions:

- A library resource can be used to specify a list of Read-only database tables that are to be read into a memory cache. Access to these tables, through the General I/O activity, is considerably faster than accessing database tables on disk.
- 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 tune the behavior of flows at execution time. For example, by modifying the value of a global variable that contains a customer risk threshold, the boundary between a medium-risk customer versus a high-risk customer can be adjusted at run time without changing any expressions or 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.

ConcurrentWait Node

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

Fault Response

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

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 optional Boolean concurrent attribute that indicates whether they should be executed concurrently. If this attribute is true, then these nodes are scheduled for execution on a thread in parallel with the main thread of execution. If the attribute is false, then the nodes execute 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:

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

If the nodes are marked concurrent, then step 2 is executed on a separate thread and the main thread continues processing the next node. Step 3, for the concurrent node, 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 method could be used for asynchronous execution.
2. The copying back of values to process variables takes place in the main execution thread. However, if the same process variables are referenced for output in other concurrent nodes, the last node is executed.

   Note: The value from the last concurrent node to finish is used. This is indeterminate behavior and is not recommended.
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.

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 contains 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.

Chapter 3

Common Operations

Promoting Decision Flows

Overview

Some SAS solutions that embed SAS Decision Services provide their own promotion frameworks, which automate the promotion 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 13.) 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 3.

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 is shipped with a rich set of activities. If your organization develops a new activity that extends SAS Decision Services functionality, that activity must be published 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.

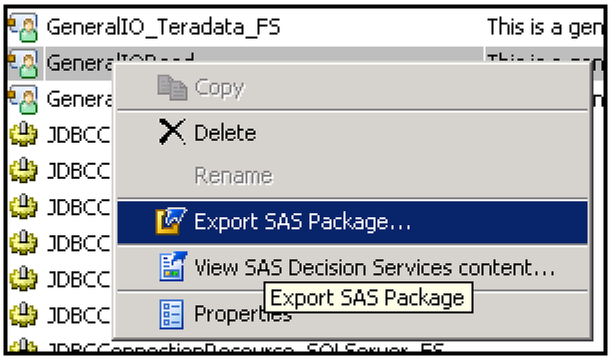
The artifact types that you can export are activity, flow, variable, event, and resource.

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.

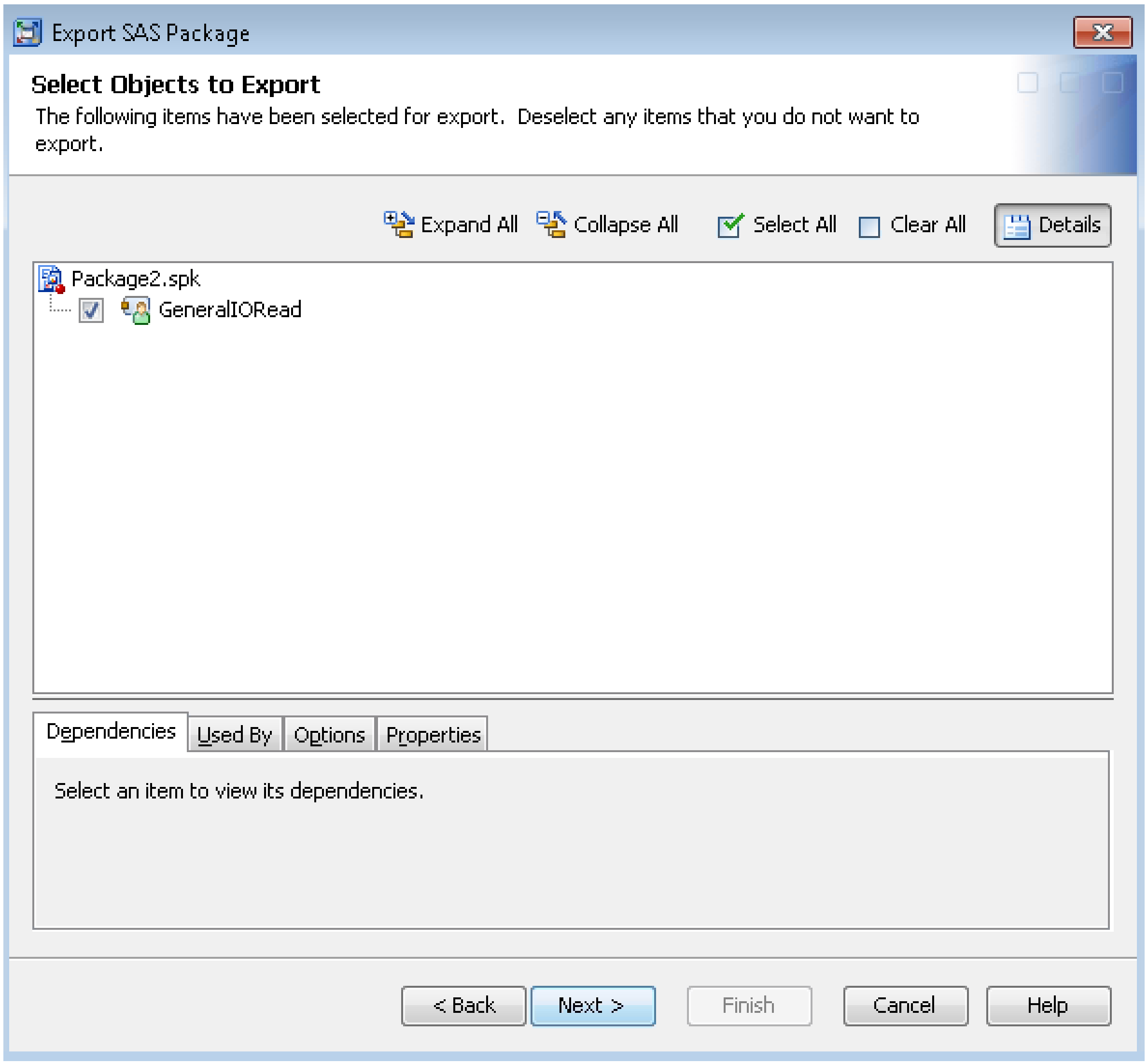
The following example illustrates the promotion of a flow from a development repository to a production repository. Although both repositories are contained by the same content mapped folder and application context in the example, these conditions are not required.

1. Launch SAS Management Console and click the **Folders** tab.
2. Expand the **`System`** folder and the **`Applications`** folder.
3. Expand the **`SAS Decision Services`** and the **`Decision Services 6.2`** folders.

4. Select **SASDSEngineRepository**.
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**.



The following items have been selected for export. Deselect any items that you do not want to export.

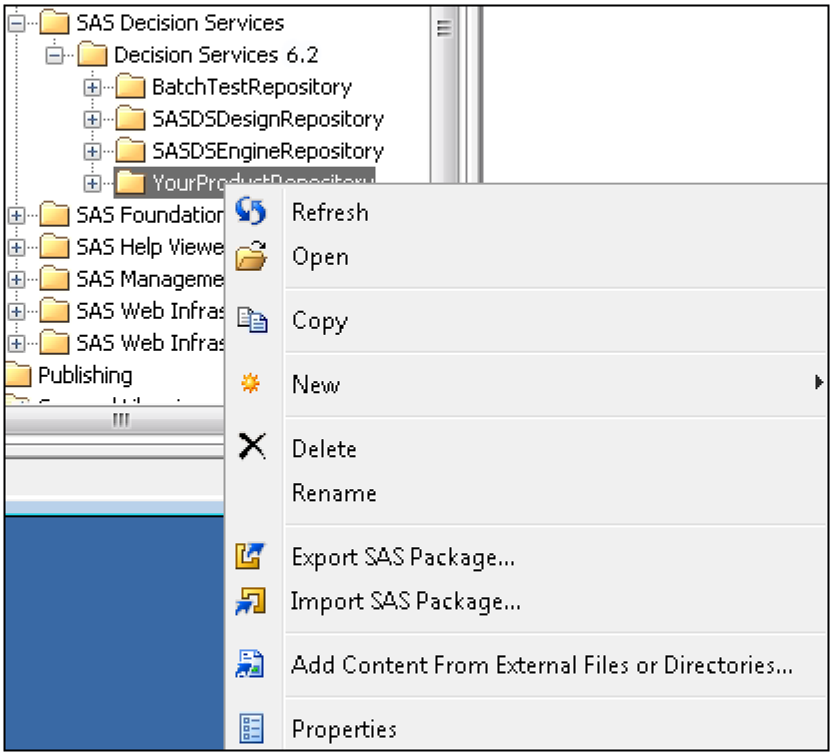
Select an item to view its dependencies.

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.

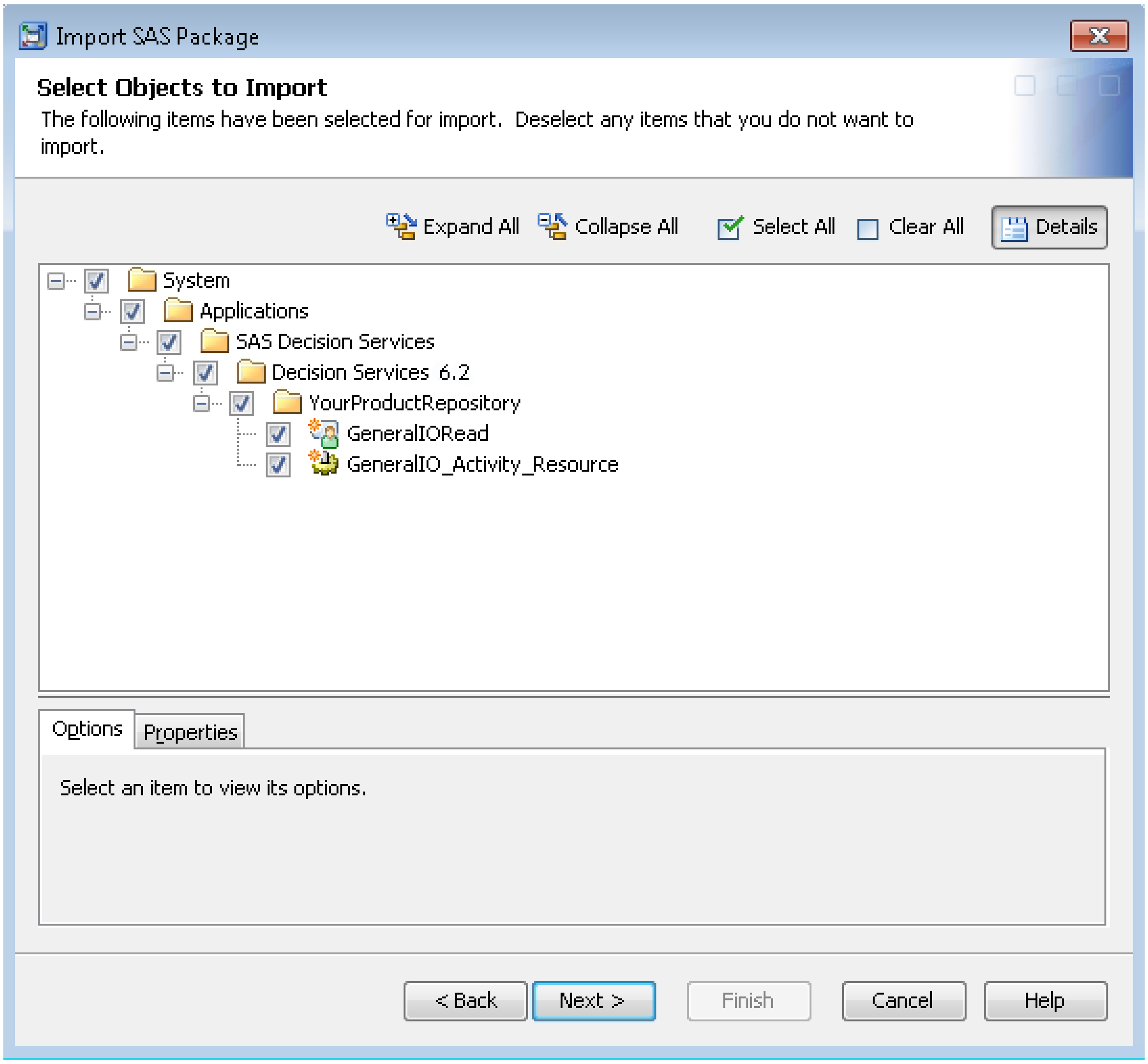
9. Right-click the repository folder of the repository that you want to promote the artifact to, and select **Import SAS Package**.

   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**.


Import SAS Package

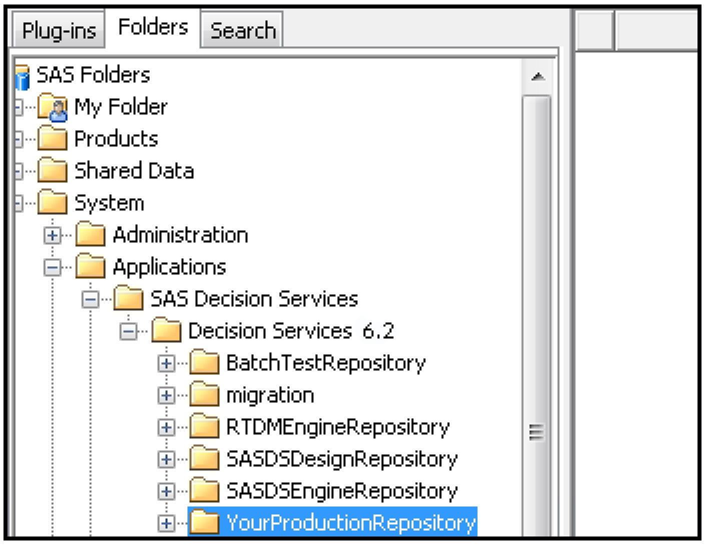
Select Objects to Import

The following items have been selected for import. Deselect any items that you do not want to import.

Select an item to view its options.

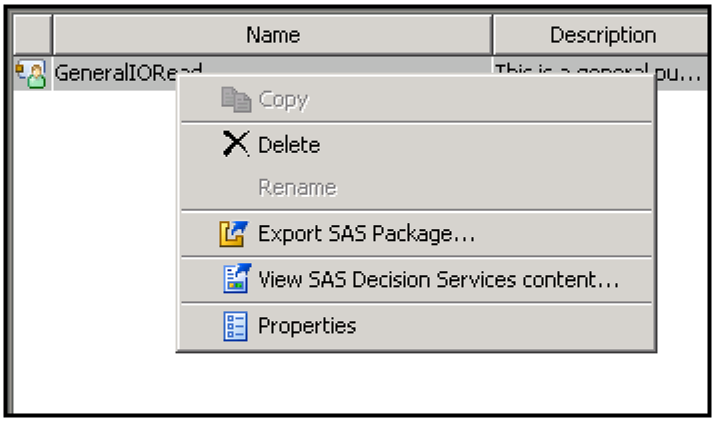
12. Verify that the summary is correct and click **Next**.
13. Click **Finish**.

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.
]

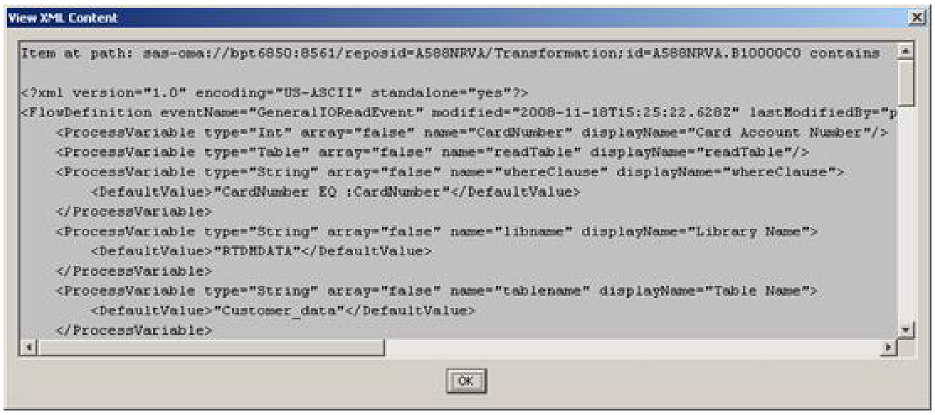


You can further verify that the promotion process was successful by viewing the contents of the XML file after promotion.

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.



```
Item at path: sas-oma://bpt6850:8561/reposid=A588NRVA/Transformation;id=A588NRVA.B10000C0 contains

<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<FlowDefinition eventName="GeneralIOReadEvent" modified="2008-11-18T15:25:22.628Z" lastModifiedBy="p
    <ProcessVariable type="Int" array="false" name="CardNumber" displayName="Card Account Number"/>
    <ProcessVariable type="Table" array="false" name="readTable" displayName="readTable"/>
    <ProcessVariable type="String" array="false" name="whereClause" displayName="whereClause">
        <DefaultValue>"CardNumber EQ :CardNumber"</DefaultValue>
    </ProcessVariable>
    <ProcessVariable type="String" array="false" name="libname" displayName="Library Name">
        <DefaultValue>"RTDMDATA"</DefaultValue>
    </ProcessVariable>
    <ProcessVariable type="String" array="false" name="tablename" displayName="Table Name">
        <DefaultValue>"Customer_data"</DefaultValue>
    </ProcessVariable>
```

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

Activating Flows

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.

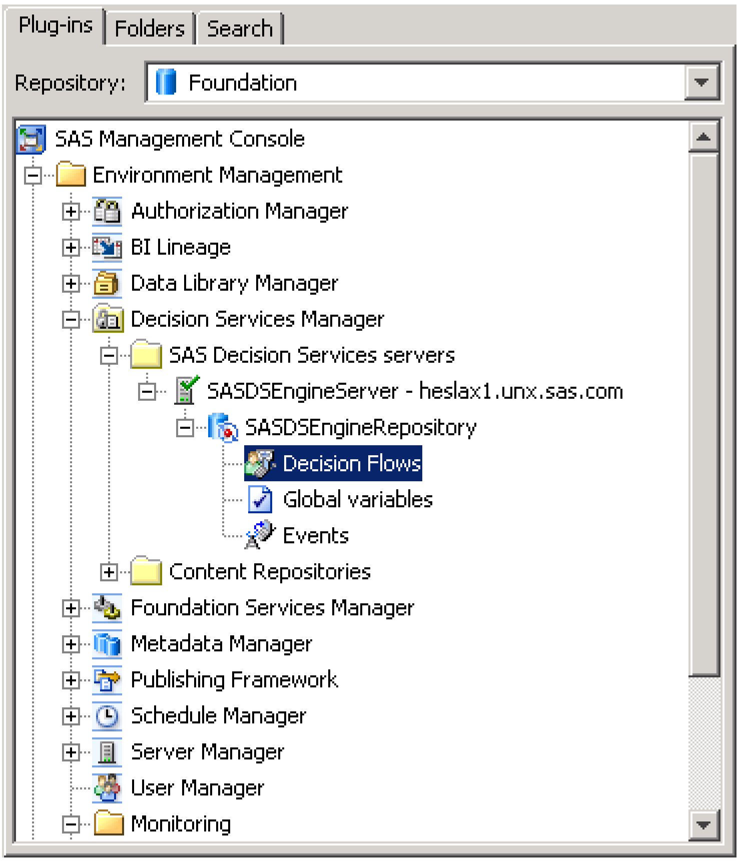
A flow is the only artifact that can be activated or deactivated. All other artifacts are used by flows, directly or indirectly, and are loaded when they are referenced by an active flow. When loaded, flows and other artifacts are synchronized across the machines in the SAS Decision Services cluster 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**.

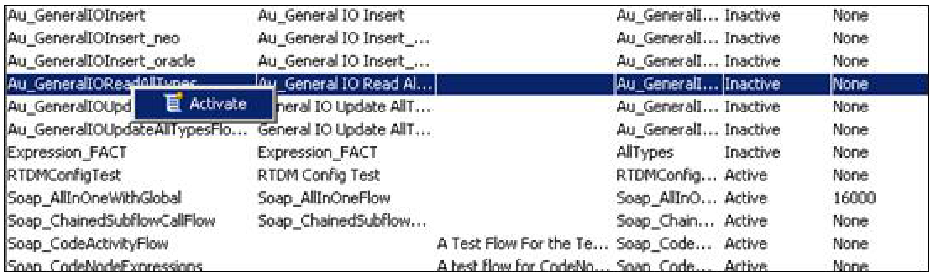


Plug-ins | Folders | Search

Repository: Foundation

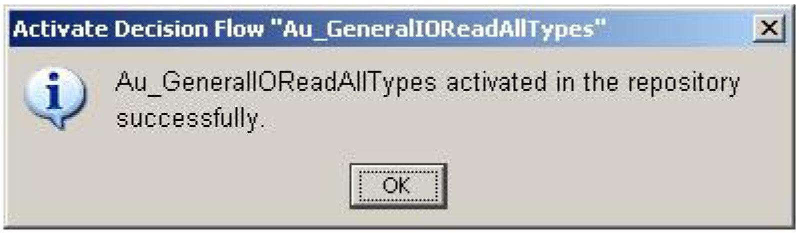
- SAS Management Console
  - Environment Management
    - Authorization Manager
    - BI Lineage
    - Data Library Manager
    - Decision Services Manager
      - SAS Decision Services servers
        - SASDSEngineServer - heslax1.unx.sas.com
          - SASDSEngineRepository
            - Decision Flows
            - Global variables
            - Events
      - Content Repositories
    - Foundation Services Manager
    - Metadata Manager
    - Publishing Framework
    - Schedule Manager
    - Server Manager
    - User Manager
  - Monitoring

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



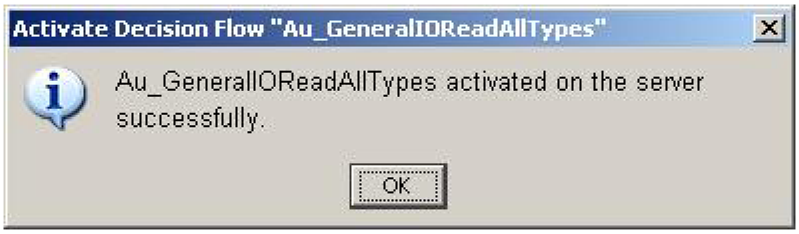
Au_GeneralIOInsert	Au_General IO Insert		Au_GeneralI…	Inactive	None
Au_GeneralIOInsert_neo	Au_General IO Insert_…		Au_GeneralI…	Inactive	None
Au_GeneralIOInsert_oracle	Au_General IO Insert_…		Au_GeneralI…	Inactive	None
Au_GeneralIOReadAllTypes	Au_General IO Read Al…		Au_GeneralI…	Inactive	None
Au_GeneralIOUpd	neral IO Update AllT…		Au_GeneralI…	Inactive	None
Au_GeneralIOUpdateAllTypesFlo…	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:



Activate Decision Flow "Au_GeneralIOReadAllTypes"

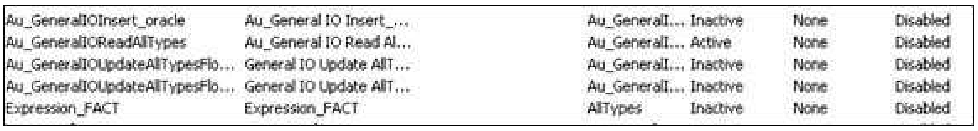
Au_GeneralIOReadAllTypes activated in the repository successfully.



Activate Decision Flow "Au_GeneralIOReadAllTypes"

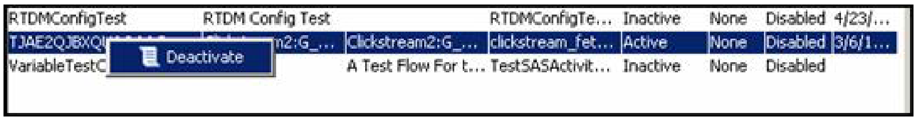
Au_GeneralIOReadAllTypes activated on the server successfully.

The first dialog box indicates that the flow was successfully marked as active in the repository. The second dialog box indicates the flow was successfully activated in the running system and is now ready to process events. The flow status changes from inactive to active, as shown below.



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

To deactivate a flow, follow the previous steps in order to view the list of flows. Then right-click an active flow, and select **Deactivate**, as shown below.



RTDMConfigTest	RTDM Config Test		RTDMConfigTe...	Inactive	None	Disabled	4/23/...
TJAE2QJBXQI	m2:G_...	Clickstream2:G_...	clickstream_fet...	Active	None	Disabled	3/6/1...
VariableTestC		A Test Flow For t...	TestSASActivit...	Inactive	None	Disabled	

Flows may be activated or deactivated in a system that is offline, to indicate which flows to load during system startup. 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 137.

Managing Global Variables

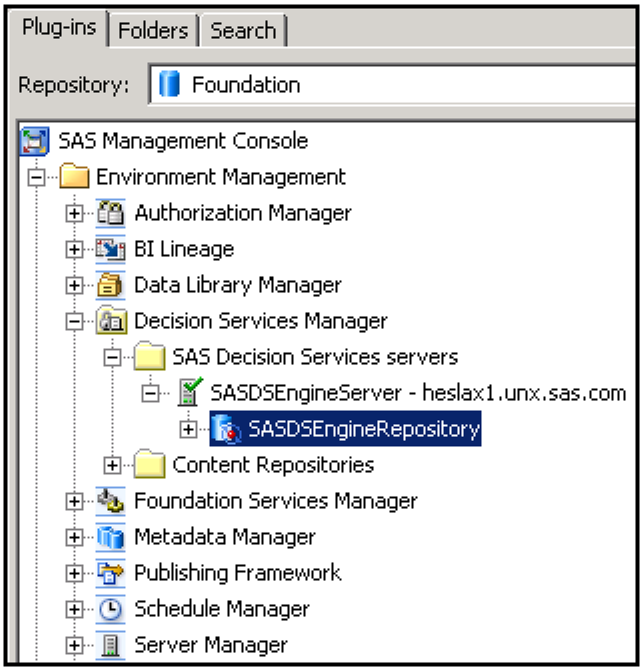
Global variables are threshold values that are used to tune the behavior of flows at execution time. Unlike process variables that are specific to a flow, the value of a global variable affects the behavior of every flow that references it.

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.

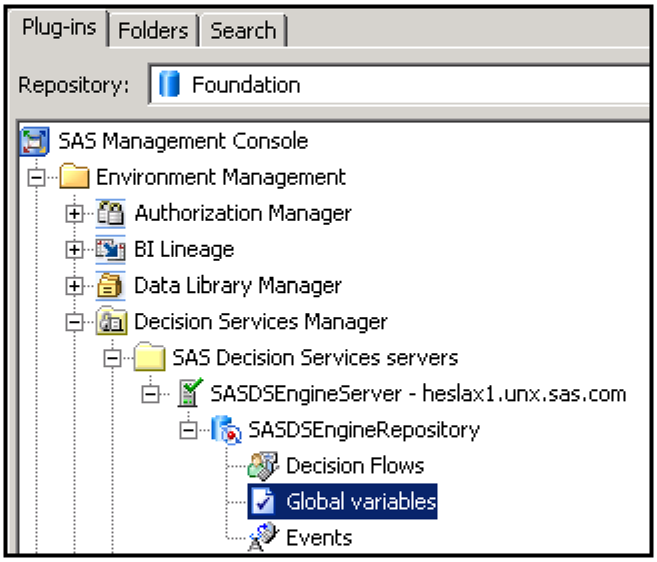
Global variables are created and assigned initial values when a flow that uses a global variable is designed. For security reasons, only an administrator whose role includes the Set Global Value capability can change the value of a global variable in a production environment.

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

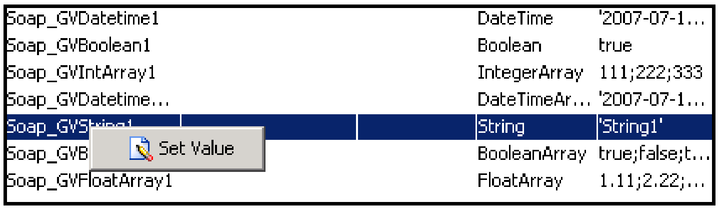
1. Launch SAS Management Console.
2. On the **Plug-ins** tab, expand `Decision Services Manager` and the `SAS Decision Services servers` folder.



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

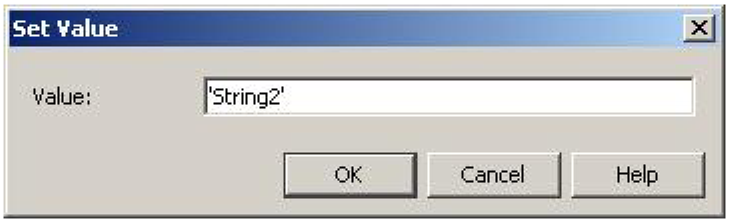


4. Right-click the global variable that you want to change, and click **Set Value**.



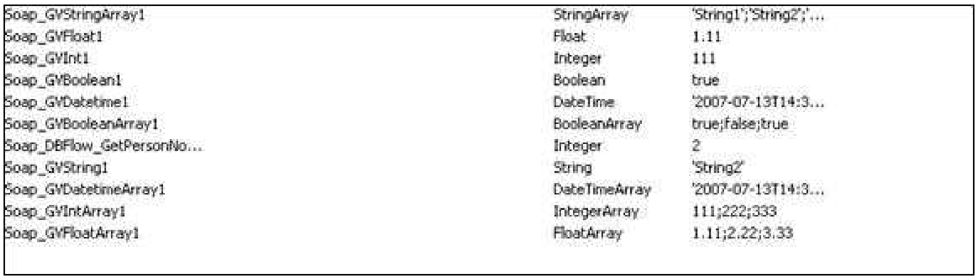
Soap_GVDatetime1	DateTime	'2007-07-1...
Soap_GVBoolean1	Boolean	true
Soap_GVIntArray1	IntegerArray	111;222;333
Soap_GVDatetime...	DateTimeAr...	'2007-07-1...
Soap_GVString1	String	'String1'
Soap_GVB	BooleanArray	true;false;t...
Soap_GVFloatArray1	FloatArray	1.11;2.22;...

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


Set Value

Value: 'String2'

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 in the SAS Decision Services enterprise archive file (EAR) during installation and configuration of the design server and engine. If no value is specified by the user, then a default value is set.
- 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. 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.

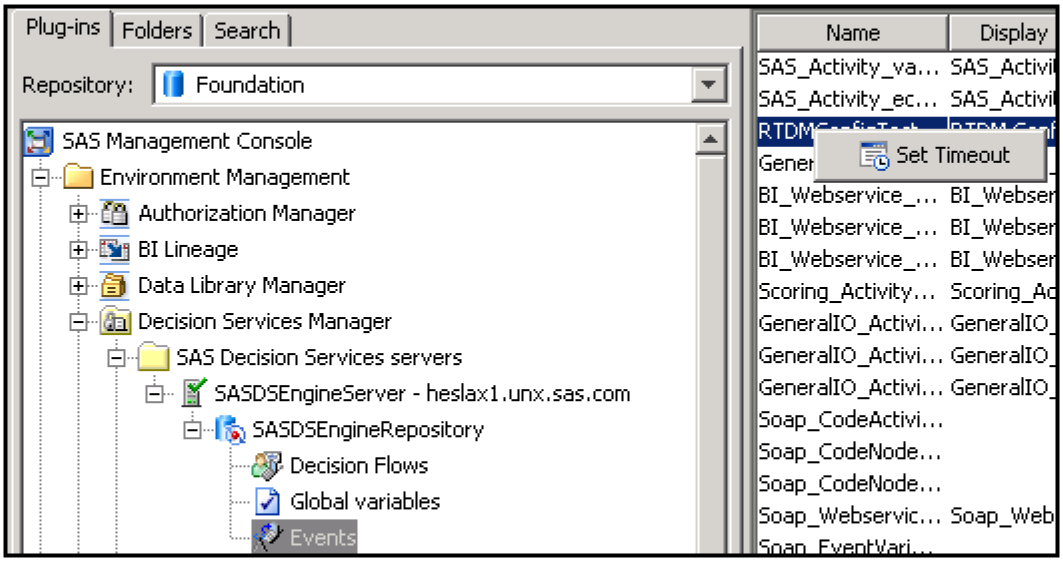
Set the event time-out value by using the Decision Services Manager plug-in for SAS Management Console. To set the time-out value for an event, follow these steps:

1. Launch SAS Management Console.

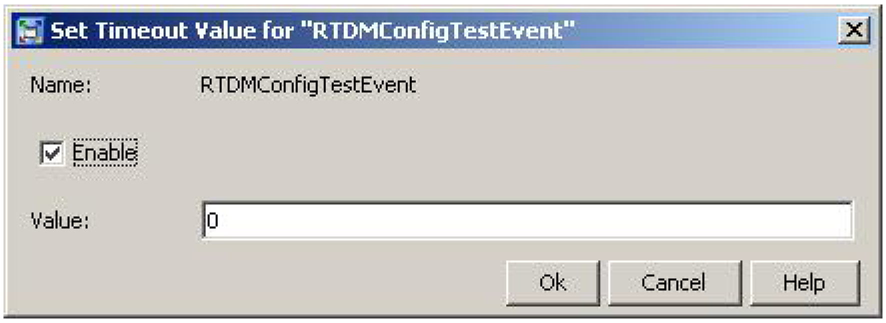
2. Expand the **`Decision Services Manager`** and **`SAS Decision Services servers`** folders.

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

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


Plug-ins
Folders
Search
Repository: Foundation
SAS Management Console
Environment Management
Authorization Manager
BI Lineage
Data Library Manager
Decision Services Manager
SAS Decision Services servers
SASDSEngineServer - heslax1.unx.sas.com
SASDSEngineRepository
Decision Flows
Global variables
Events
Name
Display
SAS_Activity_va... SAS_Activi
SAS_Activity_ec... SAS_Activi
RTDM
Gener
Set Timeout
BI_Webservice_... BI_Webser
BI_Webservice_... BI_Webser
BI_Webservice_... BI_Webser
Scoring_Activity... Scoring_Ac
GeneralIO_Activi... GeneralIO_
GeneralIO_Activi... GeneralIO_
GeneralIO_Activi... GeneralIO_
Soap_CodeActivi...
Soap_CodeNode...
Soap_CodeNode...
Soap_Webservic... Soap_Web
Soap_EventVari...

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.



Name: RTDMConfigTestEvent

☑ Enable

Value: 0

Audit Logging

Audit Logger Overview

The audit logger collects information about events that occur in the SAS Decision Services engine, and records the data in a data table. For more information, see "Tables of Audit Logging Events" on page 33. Events are logged from the SAS Decision Services engine server and from the Decision Services Manager plug-in for SAS Management Console.

These engine events are logged to data tables:

- cached flow
- cached global variables

- engine stop
- flow activate
- flow deactivate

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

Terms That Are Used in Audit Logging

The following data items are common to all events:

GUID
: A globally unique ID that is used as the primary key in order to link data in multiple tables.

Host Name
: Used to group several events from an engine server and from the Decision Services Manager plug-in.

Object Name
: A column that contains the name of a flow or of a global variable.

Object Type
: One of the following: an engine, a flow, or a global variable.

Operation
: The type of event that is being logged, such as Cached, Activate, Deactivate, and Stop.

Timestamp
: A sequence of characters that denote the date and time at which a certain event occurred.

Setting Up the Audit Logging Functionality

Overview

The components of the audit logger are configured during the installation and configuration of SAS Decision Services. The $Audit_Log_JDBCConnectionResource is created specifically for the audit logger. It points to the database that is to be used to store the audit log. During configuration, it is possible to set a schema for the audit log. If the schema is left blank, the default schema for the credentials set in the $Audit_Log_JDBCConnectionResource is used.

The data tables (that are required for audit logging) are created during installation or configuration. When SAS Decision Services is installed, no flows are active. Therefore, no entries exist that need to be cached. When a flow is activated, the corresponding event is logged. The engine then caches the flow, and that flow is another event that is logged.

Tables of Audit Logging Events

Audit logging events are recorded in the following four tables. The tables are located in the database that $Audit_Log_JDBCConnectionResource points to.

- AuditLog
- AuditLogFlows

- AuditLogGlobals
- AuditLogGlobalValues

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

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

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

Data That Is Logged for Cached Global Variable Events

The following information is logged for each cached global variables event:

- GUID
- Name
- Type
- IsArray
- Value
- Index

Data That Is Logged for Cached Flow Events

The following information is logged for each cached flows event:

- GUID
- Name
- Type

Data Collection for Performance Analysis

The SAS Decision Services User Log

Overview

The user log collects information about how specific events flow through the engine. It should be used for short periods of time only. It can also be used to debug specific events. Do not use the user log in the production environment because it has a significant performance impact.

The JDBC Connection system resource that is used by the user log is specified in the rtdm_config.properties file as rtdm.user.log.resource.name. This value is set as part of the configuration process.

What the SAS Decision Services User Log Contains

The user log contains several XML documents. Each document has a top level element called TestOutput. Each XML document represents one decision flow invocation and contains all of the following information:

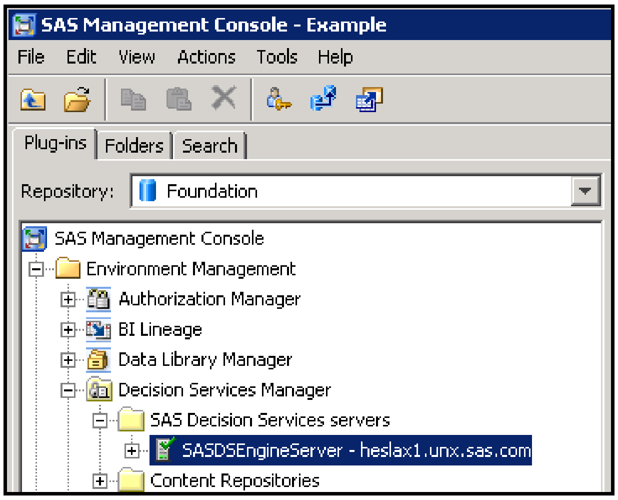
- the event request data
- the values of the process variables before executing each activity
- the values of the process variables after executing each activity
- the path that the event traveled through the flow

Location of the User Log

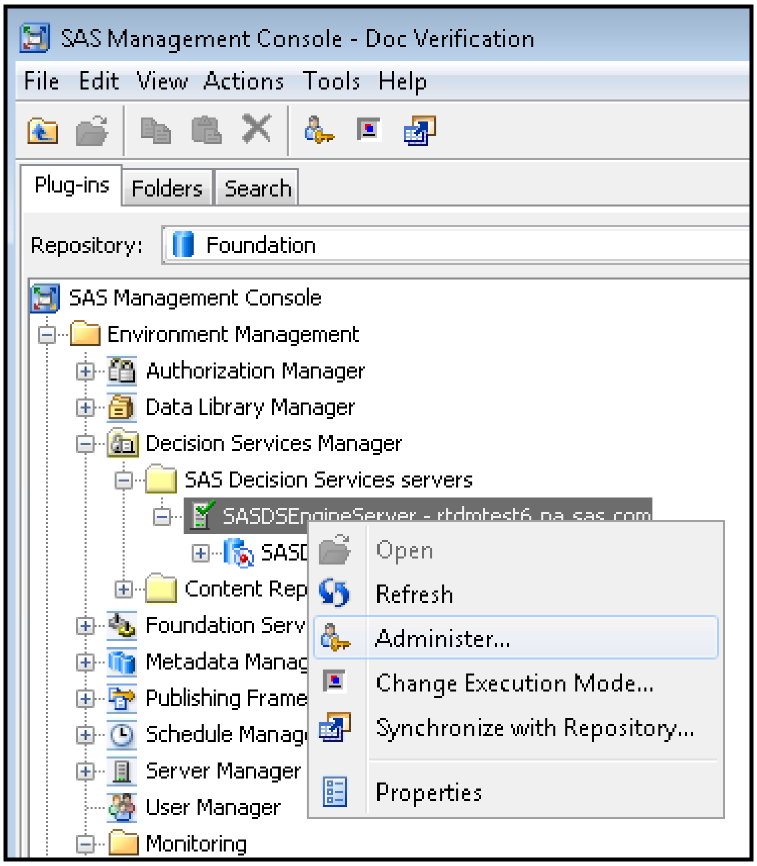
The location of the user log is set during the installation and configuration process of SAS Decision Services. The location is the database that is indicated in the $User_Log_JDBCConnectionResource. During configuration, it is possible to set a schema for the user log. If no schema is specified, the default schema for the credentials that were set in the $User_Log_JDBCConnectionResource is used.

To enable the user log, follow these steps:

1. Launch SAS Management Console.
2. Expand **`Decision Services Manager`** and **`SAS Decision Services servers`**.

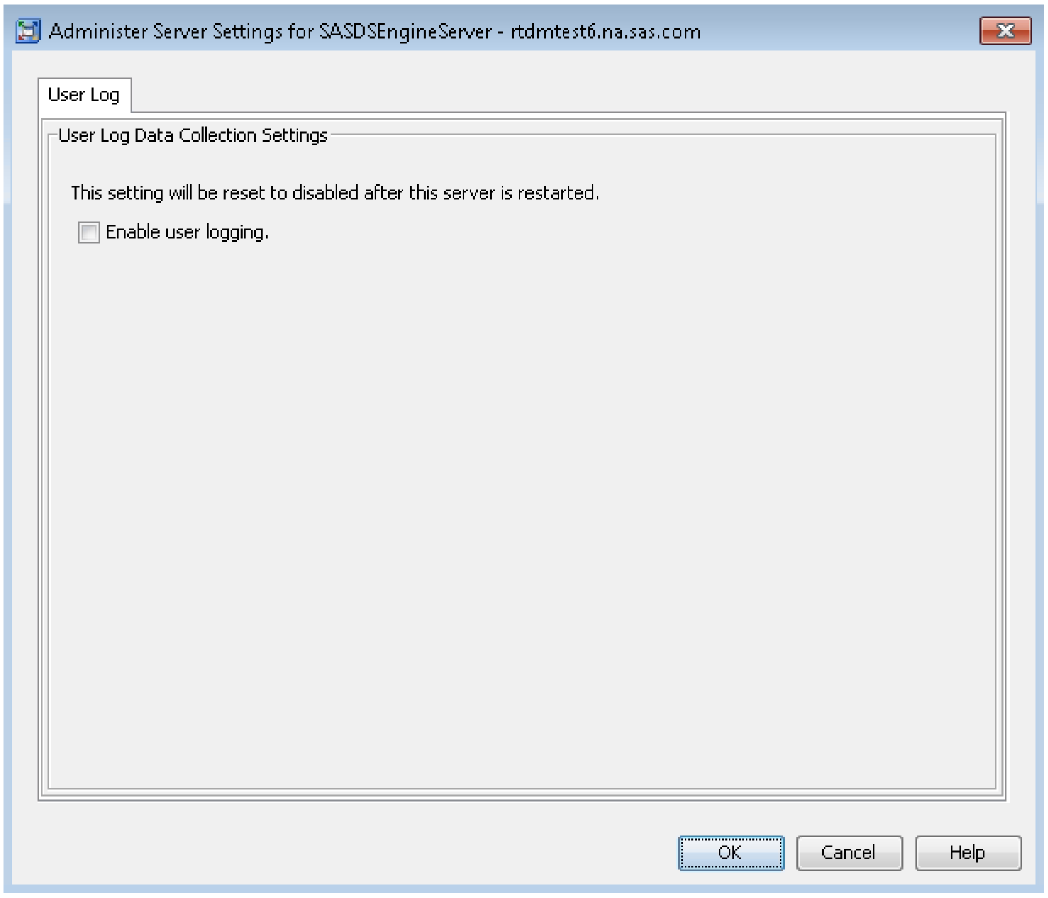


3. Right-click the system that you want to collect performance data for, and select **Administer**.



4. On the **User Log** tab, select the check box for **Enable user logging**.

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



Administer Server Settings for SASDSEngineServer - rtdmtest6.na.sas.com

User Log

User Log Data Collection Settings

This setting will be reset to disabled after this server is restarted.

☐ Enable user logging.

Chapter 4
Environments and Resources

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.

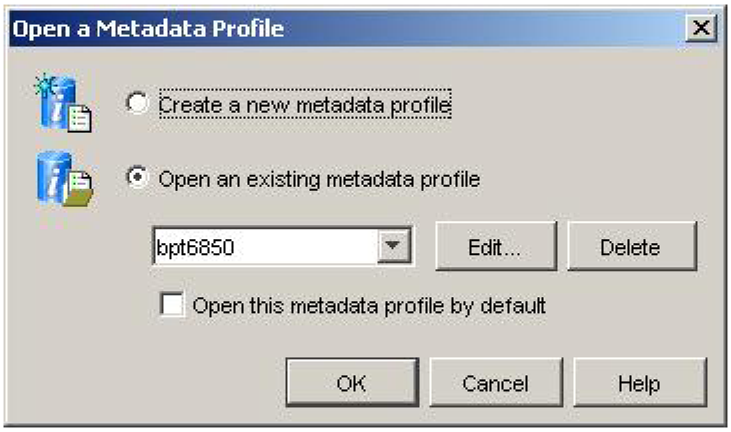
A repository does not have to be associated with a server; it can be used simply as a storage area for artifacts.

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

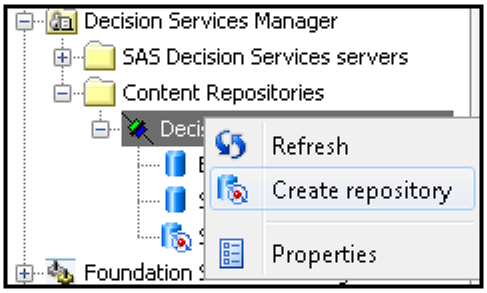
To create a new SAS Decision Services repository, follow these steps:

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

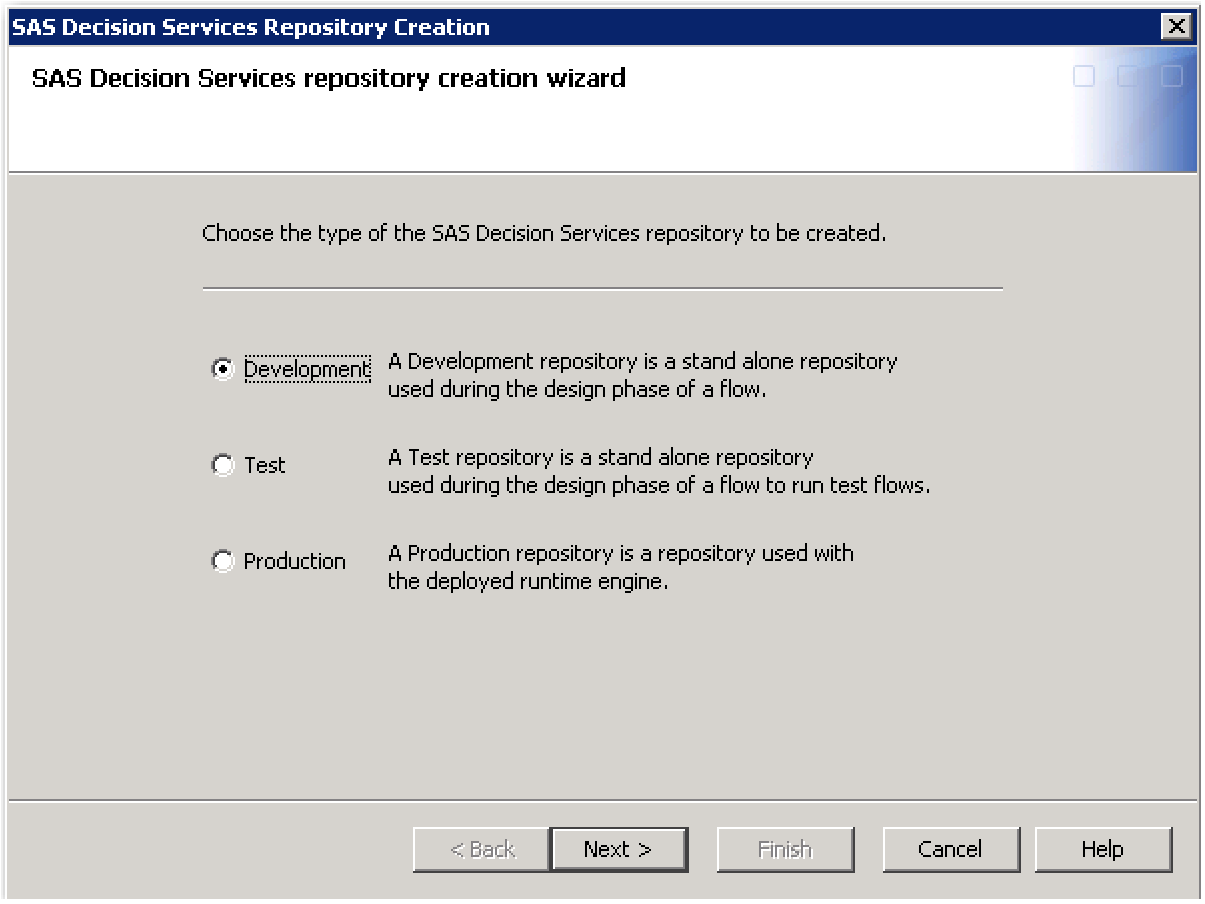

Open a Metadata Profile
Create a new metadata profile
Open an existing metadata profile
Edit...
Delete
Open this metadata profile by default
OK
Cancel
Help

Open an existing metadata profile: bpt6850

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



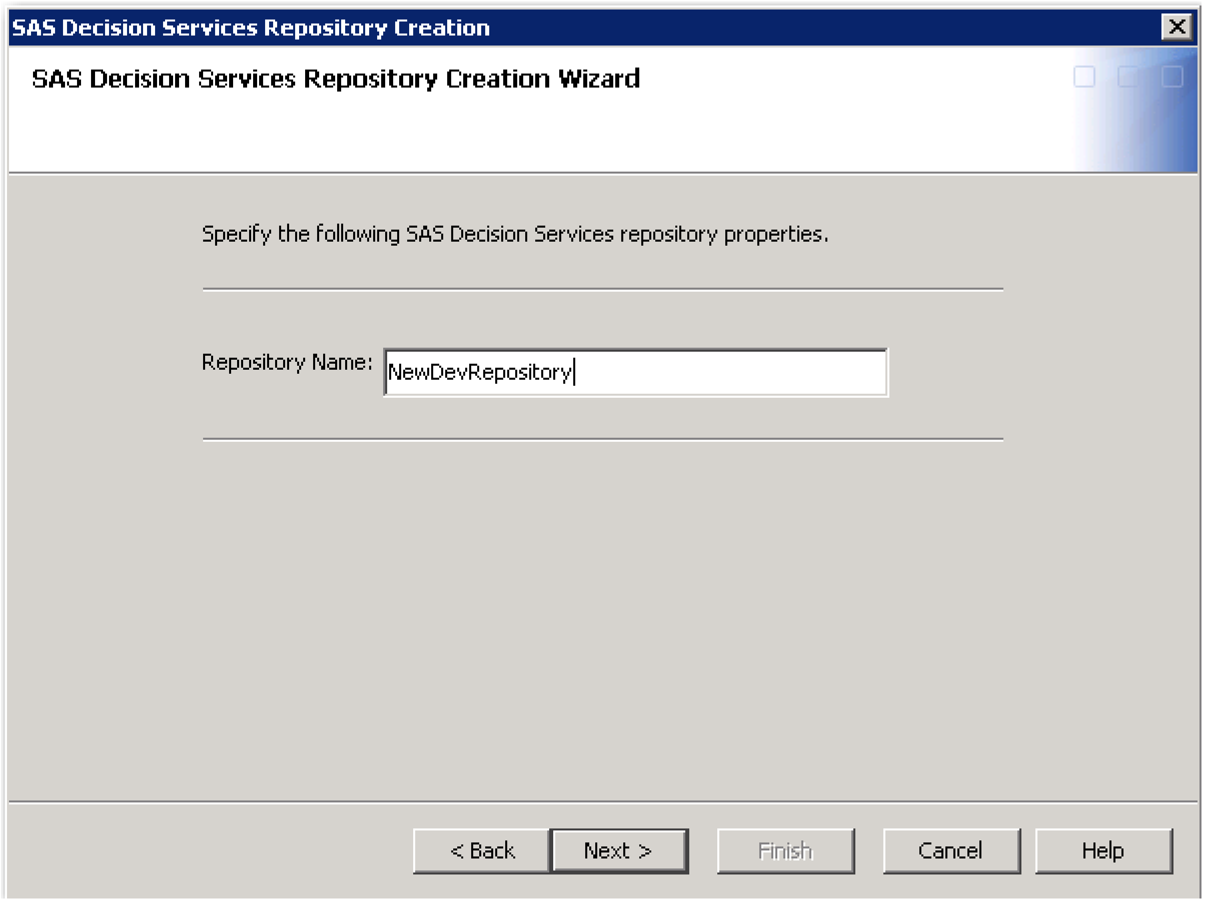
4. Choose either a development, test, or production repository. Click **Next**.


SAS Decision Services Repository Creation
SAS Decision Services repository creation wizard

Choose the type of the SAS Decision Services repository to be created.

[✓ Development]	A Development repository is a stand alone repository used during the design phase of a flow.
Test	A Test repository is a stand alone repository used during the design phase of a flow to run test flows.
Production	A Production repository is a repository used with the deployed runtime engine.

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

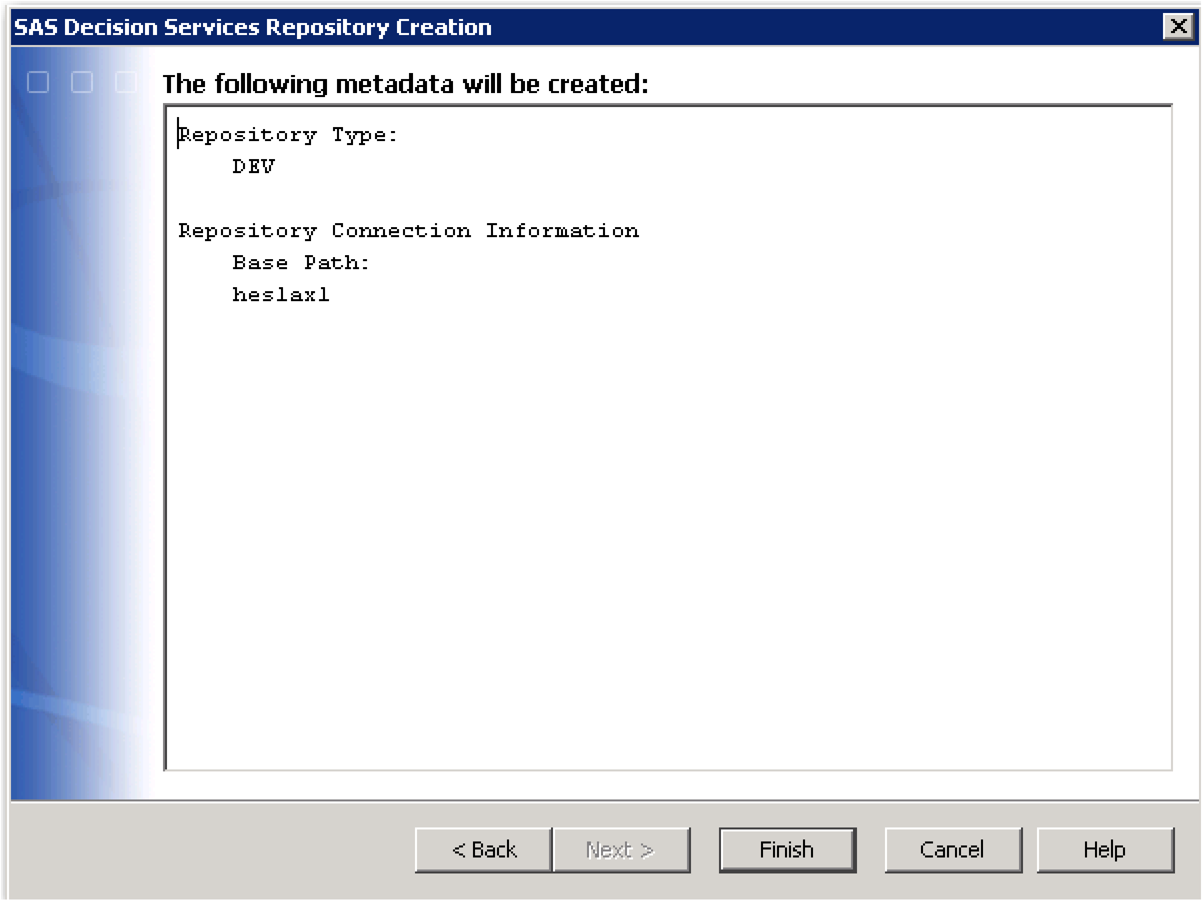

SAS Decision Services Repository Creation

SAS Decision Services Repository Creation Wizard

Specify the following SAS Decision Services repository properties.

Repository Name: NewDevRepository

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



The following metadata will be created:

```
Repository Type:
    DEV

Repository Connection Information
    Base Path:
    heslaxl
```

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

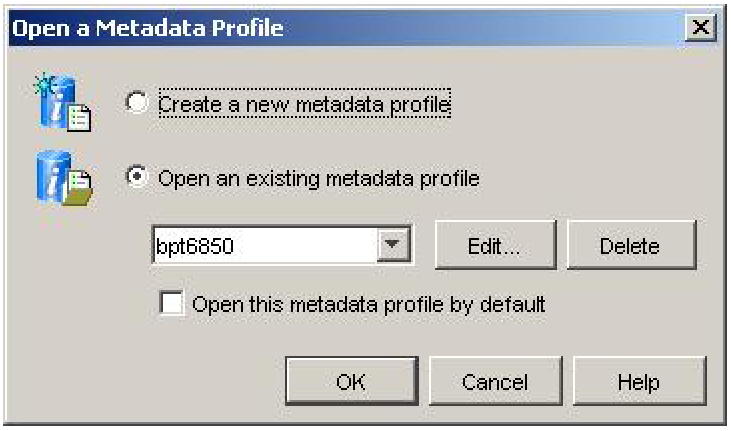
A repository is bound to an engine or design server when that server is installed and configured. For more information, see Chapter 7, “Installation,” on page 87.

Delete a Repository

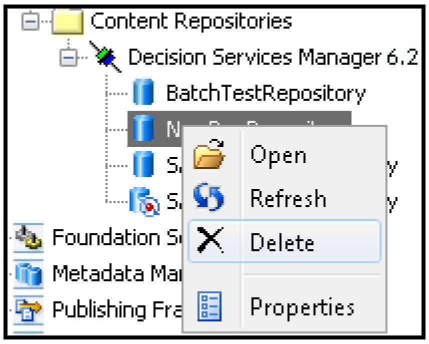
CAUTION:
Deleting a repository is an irreversible operation.

To delete a repository:

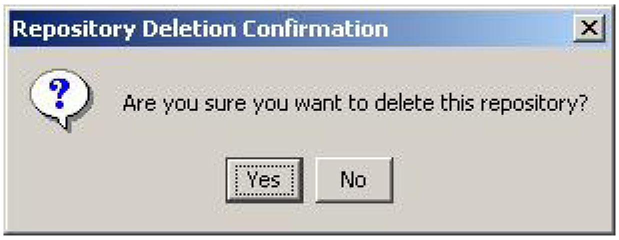
1. Log on to SAS Management Console. Choose the metadata profile that is associated with the SAS Metadata Repository that contains the repository to delete.


Open a Metadata Profile
Create a new metadata profile
Open an existing metadata profile
bpt6850
Edit...
Delete
Open this metadata profile by default
OK
Cancel
Help

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**.


Repository Deletion Confirmation
Are you sure you want to delete this repository?
Yes
No

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, the JDBC Connection system resource is used to connect to database servers for use in the General I/O activity. These resources point directly to the database using the database's own JDBC driver.

The web service system resource is 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.

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 46. In the case of General I/O, the Connection Options value is not required.

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 by default.

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 104.

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 load balances every SAS Federation Server. Therefore, 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 excess processing capacity available to all processes. It also maximizes the retention of processing capacity in the event of a server failure.

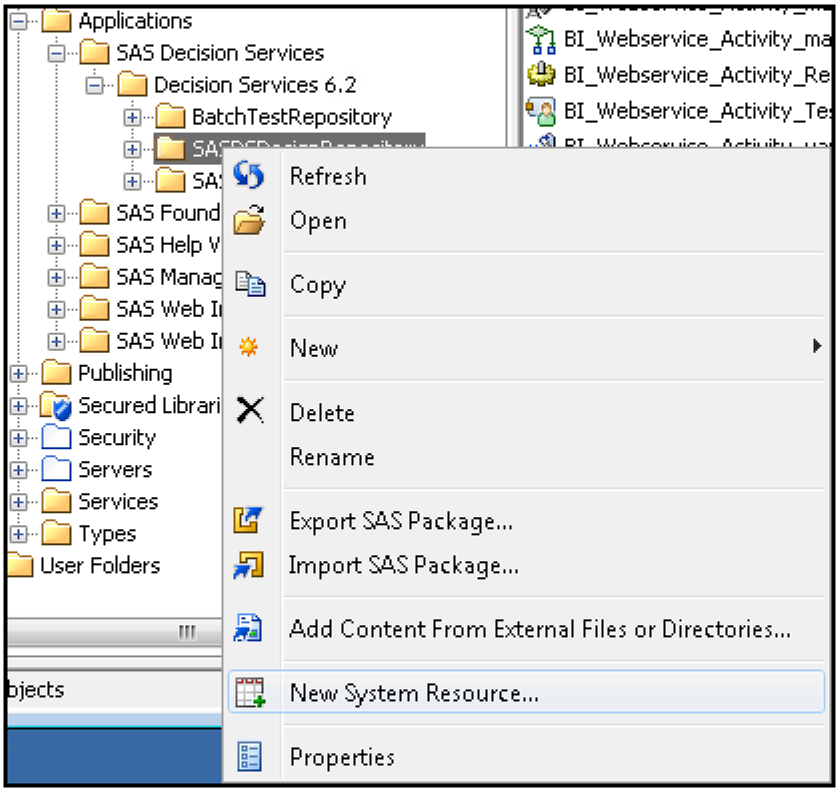
The following activity types use the JDBC Connection system resource:

- SAS Activity
- General I/O Activity

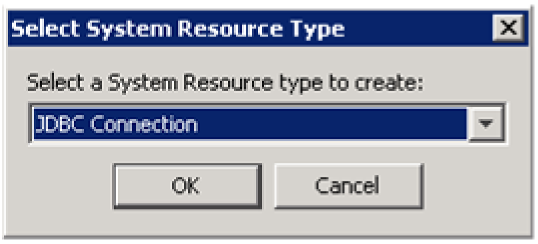
Specify a New System Resource as a JDBC Connection

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

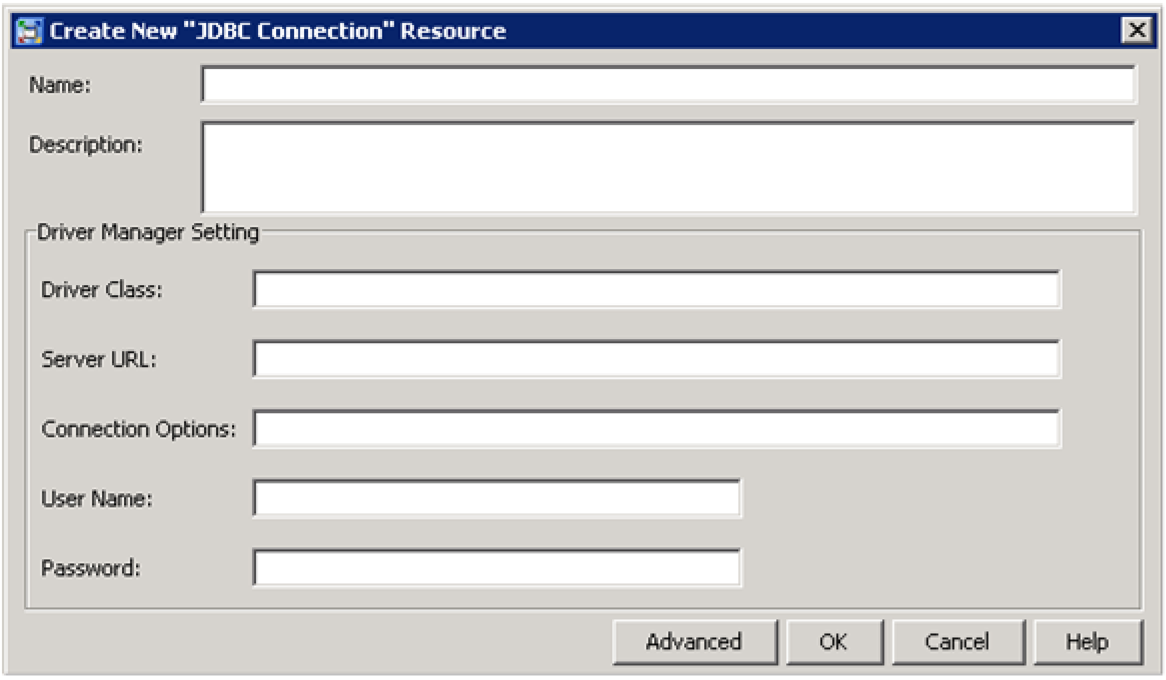
1. Expand **System** ⇨ **Applications** ⇨ **SAS Decision Services** ⇨ **Decision Services 6.2**.
2. Right-click a repository folder such as **SASDSDesignRepository**.
3. Select **New System Resource** from the drop-down menu.



4. Select **JDBC Connection**.


Select System Resource Type
Select a System Resource type to create:
JDBC Connection
OK
Cancel

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 *Examples for the Driver Class Field*

Database	**Class Name**
Oracle	oracle.jdbc.driver.*OracleDriver*
SQL Server	com.microsoft.sqlserver.jdbc.*SQLServerDriver*
Teradata	com.teradata.jdbc.*TeraDriver*
DB2	com.ibm.db2.jcc.*DB2Driver*

Note: To create a system resource for accessing a SAS Federation Server, enter **`com.sas.tkts.TKTSDriver`**.

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:@//[*OraclePath*]
SQL Server	jdbc:sqlserver://[*SQL Server Host*]
Teradata	jdbc:teradata://[*Teradata Server Host*]
DB2	jdbc:db2:[*DB2 Server*]

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 104 , for more information.

Specify a New System Resource as a Web Service Connection

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

To specify a Web Service Connection as a system resource, follow steps 1–3 in "Specify a New System Resource as a JDBC Connection" on page 46, 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 46, 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 46, 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 that you created above from the drop-down list.

(Optional) Specify Tables to Cache in Memory

SAS Decision Services provides a memory cache for hosting read-only tabular data. This is an optional, performance-enhancing feature. Data in memory can be accessed much faster than data on disk. Good candidates for caching are tables that change very infrequently, but are referenced frequently. For example, a table of automobile part numbers, names, and descriptions would be a good candidate.

To specify tables to be cached, create a library resource for the schema that contains the table, and fill in the fields under **Cached Tables**. Add a row for each table to be cached in memory. You must create a library resource for each schema that contains tables to cache.

Note: Sufficient memory must exist on each middle-tier engine server to hold all tables specified for caching. Otherwise, run-time errors result. Large tables should not be cached.

Table Name
: name of table to be cached in memory.

Columns Clause
: a comma-separated list of columns to cache, or * for all columns.

Order By Clause
: (optional) a comma-separated list of columns to sort the in-memory table by.

Cached
: if checked, the table is cached. Otherwise, the table is not cached.

If you do not want to create an alias for the name of the schema that contains tables to cache, enter the same value in both the **`Name`** and **`Schema Name`** fields.

Databases

Overview

SAS Web Infrastructure Platform Data Server is included in your deployment for use as transactional storage by SAS Decision Services Monitor software. The server is based on PostgreSQL 9.1.9. 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 Monitor 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 Monitor 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
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.

Chapter 5
Decision Services Activities

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.

SAS Decision Services uses the following classifications of configurable activities:

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

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 file that describes the activity, the methods that are supported by that activity, and the system resources that are used by that activity.

SAS DATA STEP 2 (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. 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 does not support multiple methods per activity, so execute() is the only method that is called. 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 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, 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* and *SAS 9.4 DS2 Language Reference: Getting Started*, available at **`http://support.sas.com/documentation/solutions/ds2/`**.

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(varchar(32767) in_string, in_out varchar 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 SAS Federation 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.

   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 methods that are exposed by your activity. 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_NULL 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, you must notify any running engines that the activity has changed. Existing SAS Federation Server connections in a running engine continue to use the original activity until you have reset the SAS Federation Server connections or restarted the engine. To reset the connections, go to the Decision Services Manager plug-in in SAS Management Console, and perform the following steps:
   a. On the **Plug-ins** tab, select **Environment Management** ⇨ **Decision Services Manager** ⇨ **SAS servers**.
   b. Right-click the engine server object, and select **Reset Federation Server Connections**.
   c. Confirm that you want to reset the SAS Federation Server connections.

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 \SASDecisionServicesServerConfig6.2\SASCode`**. Some of those SAS files are utilities to help you build your own SAS activities, and some are sample SAS activities.

Utility Packages

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_logger

This package provides a basic logging capability. By using this package, the DS2 package author can write messages to the SASDS_xxx_yyy.log file that SAS Federation Server points to. In the log file, xxx is the current system date, and yyy is the process ID. If desired, the DS2 package author can specify a logger target other than the default. Doing this allows for setting different logging levels and destinations for different loggers. Whether a message is printed to the log depends on the logging level that is set in the log configuration file.

Use the following to instantiate a tap_logger:

- declare package tap_logger logger(); - Uses the default target of App.SASDS
- declare package tap_logger logger(varchar logger_name); - Uses the target passed in by caller

Here are the available logging methods:

- trace(varchar msg); - Log a trace level message.
- debug(varchar msg); - Log a debug level message.
- info(varchar msg); - Log an information level message.
- warn(varchar msg); - Log a warning level message.
- err(varchar msg); - Log an error level message.
- fatal(varchar msg); - Log a fatal level message.
- isTraceEnabled() returns int; - Returns 1 if enabled, 0 otherwise.
- isDebugEnabled() returns int; - Returns 1 if enabled, 0 otherwise.
- isInfoEnabled() returns int; - Returns 1 if enabled, 0 otherwise.
- isWarnEnabled() returns int; - Returns 1 if enabled, 0 otherwise.
- isErrEnabled() returns int; - Returns 1 if enabled, 0 otherwise.

- isFatalEnabled() returns int; - Returns 1 if enabled, 0 otherwise.

You can also use the tap_logger package to debug your code, as well as to check the output in the SAS Decision Services log on SAS Federation Server. The configuration for the SAS Federation Server log is located in ***Federation Server Install Root*\etc\dfs_log.xml**. The logs themselves are located in ***Federation Server Install Root*\var\logs**. To take advantage of the tap_logger package, edit the dfs_log.xml file. First, add an entry for a new file appender for the SASDS log:

```
<!-- Rolling log file with default rollover of midnight -->
<appender class="RollingFileAppender" name="DSTimeBasedRollingFile"
   name="DSTimeBasedRollingFile">
      <param name="Append" value="true"/>
      <param name="ImmediateFlush" value="true"/>
      <rollingPolicy class="TimeBasedRollingPolicy">
         <param name="fileNamePattern"
          value="[Federation Server Install Root]\var\log\SASDS_%d_%S{pid}.log"/>
      </rollingPolicy>
      <layout>
         <param name="HeaderPattern" value="Host: '%S{hostname}',
           OS: '%S{os_family}', Release: '%S{os_release}',
                SAS Version: '%S{sup_ver_long2}',
           Command: '%S{startup_cmd}'"/>
         <param name="ConversionPattern" value="%d %-5p [%t] %c
          %X{Client.ID}:%u - %m"/>
      </layout>
   </appender>
```

Note: Note: [Federation Server Install Root] must be set to the correct path for the given installation and logging level. The level value can be set to Info, Warn, Debug, Error, or Trace, as desired.

Next, add a logger entry that uses the new file appender. Set the logging level to the desired level:

```
<logger name="App.SASDS">
   <level value="Info"/>
     <appender-ref ref="DSTimeBasedRollingFile"/>
</logger>
```

Here the default logger target name of App.SASDS is used. When a custom log target was is used (for example

```
declare package tap_logger logger('customLog');
```

), you must still add a file appender in the same manner as before. However, be sure to use a unique value in the name field. Then, add the logger, using the same name as the one that you used when the tap_logger package was declared:

```
<logger name="customLog">
      <level value="Info"/>
        <appender-ref ref="customLogFile"/>
   </logger>
```

Note: Any tap_package that is instantiated with the name 'customLog' will write its log messages to the appender that is pointed to by this entry in the dfs_log.xml file. This feature allows for multiple log targets, each with their own logging level, to be written to different files. As a result, the debugging of DS2 programs is easier. Logging levels of trace, debug, info, warn, and error are available. A logger can use the same appender if it is desired that each logger have its own logging level. As an alternative, each can have a separate appender to separate the logging messages.

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_sqltable

Note: The tap_table package has been replaced by tap_sqltable in SAS Decision Services 6.2. However, support for tap_table has been retained for backward compatibility. For information about tap_table, see *SAS Decision Services 5.6 Administrator's Guide*. For new development, use tap_sqltable.

SAS Decision Services table objects are passed to a DS2 method as an encoded string (varchar) parameter. To decode the string, you must use the tap_table package. Empty table objects can also be created and populated in your custom SAS activity code. This package provides an encode() method that can be called to create an encoded string version of the current table that is returned to the SAS Decision Services engine.

Note: There is a 32K limitation on the size of string (varchar) that can be passed to a DS2 method. The size of an encoded table depends on the sizes and numbers of the data types that it contains. Additional space is used by record and field separators. Therefore, very large tables should be avoided or broken up into multiple smaller tables.

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.
- delete_column(varchar name); - Removes the given column from the table.
- delete_row(); - Removes the given row from the table.
- 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.

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.
- 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, int 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 int 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.

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.

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;
```

For more information, see the *SAS 9.3 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 48.

Invoking SAS BI Web Services

SAS BI web services executes as SAS stored processes in the SAS server tier. 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:

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	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	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

Celebrus Real-Time 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 Celebrus Real-Time Engine server using an HTTP transport.

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

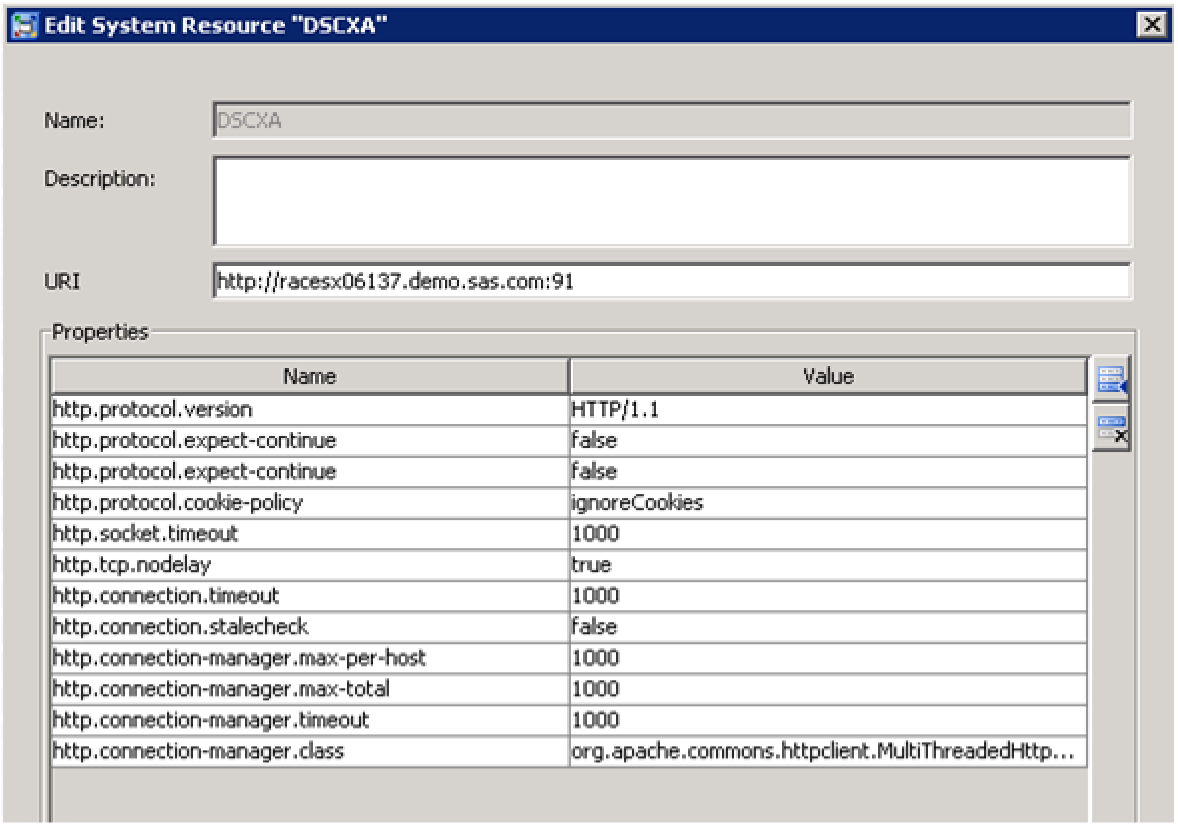
As with other activity and resource pairs, there is a separation of workload. HttpResource is responsible for defining the location of the Celebrus Real-Time Engine server, as well as how to reach it and system parameters that provide the most efficient data exchange and throughput. These values usually vary for different installations and deployment. 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 6.2`**, 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 Celebrus Real-Time Engine server. 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 3.1. 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.



Name:	DSCXA
Description:	
URI	http://racesx06137.demo.sas.com:91

Properties

Name	Value
http.protocol.version	HTTP/1.1
http.protocol.expect-continue	false
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 available 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.

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.

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: SCUpdateTable.

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.

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.

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.

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.

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 - 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.

If table caching is desired, a library resource must be used to specify cached tables.

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.

Middle-Tier Code Activities

Overview

Note: Code activities can be created only programmatically. Solutions such as SAS Real-Time Decision Manager use code activities to perform various functions. Because a user cannot create code activities directly, this section is provided for information purposes only.

Code activities execute entirely within the SAS Decision Services engine as inline code within the flow control logic.

Here is an example of a code activity in XML format:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<ActivityDefinition
      javaClassName="com.sas.analytics.ph.rt.act.code.CodeActivity"
timeout="0"
      displayName="Code Activity Test"
name="CodeActivityVariableTest"
      xmlns="http://www.sas.com/xml/analytics/rdm-1.1">
     <Description/>
      <Method displayName="sc_variable_test"
name="sc_variable_test">
    <Description>This Method Tests All the different
                 Variable types</Description>
           <Body>
                  <Expression>
                       MyInt = MyInt + 2;
                       MyFloat = MyFloat + 1.11;
                       MyBoolean = NOT MyBoolean;
```

```
                        MyString = Reverse(MyString);
                        MyDate = intnx('DTSECOND',MyDate,24*60*60);
                        arraySize = DIM(MyStringArray);
                        do index=1 to (arraySize / 2);
                              temp = MyStringArray[index];
                              MyStringArray[index] =
MyStringArray[arraySize - index + 1];
                              MyStringArray[arraySize - index + 1] =
temp;
                        end;
                  </Expression>
            </Body>
            <InputParameter array="false" type="Int" displayName="An
Input Integer" name="MyInt">
                  <Description>An Integer Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="Float"
displayName="An Input Float" name="MyFloat">
                  <Description>A Float Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="DateTime"
displayName="An Input Date" name="MyDate">
                  <Description>A Date Parameter in the form yyyy-MM-
dd</Description>
           </InputParameter>
            <InputParameter array="false" type="Boolean"
displayName="An Input Boolean" name="MyBoolean">
                  <Description>A Boolean (true-false)
Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="String"
displayName="An Input String" name="MyString">
                  <Description>A String Parameter</Description>
            </InputParameter>
            <InputParameter array="true" type="String"
displayName="inputStringArray" name="MyStringArray">
                  <Description>An Array of Strings</Description>
            </InputParameter>
            <InputParameter array="false" type="Int" displayName="An
Input Integer" name="arraySize">
                  <Description>An Integer Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="Int" displayName="An
Input Integer" name="index">
                  <Description>An Integer Parameter</Description>
            </InputParameter>
            <InputParameter array="false" type="String"
displayName="An Input String" name="temp">
                  <Description>A String Parameter</Description>
            </InputParameter>
            <OutputParameter array="false" type="Int"
displayName="An Output Integer" name="MyInt">
           <Description>The result of inputInt + 2</Description>
            </OutputParameter>
            <OutputParameter array="false" type="Float"
displayName="An Output Float" name="MyFloat">
```

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

User-Defined Functions

Note: Similar to code activities, user-defined functions can be created only programmatically. They provide client solution developers with a convenient means of authoring reusable functions. This section is provided for information purposes only.

User-defined functions (UD functions) are defined as *methods* in code activities. They can be called through code activity or through a regular function call in any SAS Decision Services control language expression.

The new reserved Out parameter, “_RETURN_VALUE”, accommodates functions that return a value.

In and Out parameters are defined by using the same parameter name or type in both In and Out parameter lists. When called through a regular function call, the variable in the caller changes value as if it had been passed by reference.

The ANY type can be used for input parameters to code activities. As with other types, it can be an array or scalar. The UD function is responsible for checking the type of parameters.

Note: The ANY data type is not supported outside inputs to code activity.

UD functions are loaded based on function calls in active flows. All methods from an activity are loaded, even if only one is called. If there is a compile error in any method, all functions in the entire activity are rejected. Function name resolution during the compilation of methods is independent from the order of loading of activities.

Here are example activities. Each one calls a method in the other. Each one illustrates a function with a return value, and a function with an In or Out parameter.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
- <ActivityDefinition
     javaClassName="com.sas.analytics.ph.rt.act.code.CodeActivity"
  timeout="0" name="UDFunctionsCode1"
  xmlns="http://www.sas.com/xml/schema/sas-svcs/rtdm-1.1">
- <Method name="yes">
- <Body>
  <Expression>RETURN true;</Expression>
  </Body>
  <OutputParameter array="false" type="Boolean" name="_RETURN_VALUE" />
  </Method>
- <Method name="indirectCall">
- <Body>
  <Expression>RETURN UDFunctionsCode2.yes();</Expression>
  </Body>
  <OutputParameter array="false" type="Boolean" name="_RETURN_VALUE" />
  </Method>
  </ActivityDefinition>
```

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
- <ActivityDefinition
     javaClassName="com.sas.analytics.ph.rt.act.code.CodeActivity"
  timeout="0" name="UDFunctionsCode2"
  xmlns="http://www.sas.com/xml/schema/sas-svcs/rtdm-1.1">
- <Method name="yes">
- <Body>
  <Expression>RETURN true;</Expression>
  </Body>
  <OutputParameter array="false" type="Boolean" name="_RETURN_VALUE" />
  </Method>
- <Method name="indirectCall">
- <Body>
  <Expression>RETURN UDFunctionsCode1.yes();</Expression>
  </Body>
  <OutputParameter array="false" type="Boolean" name="_RETURN_VALUE" />
  </Method>
  </ActivityDefinition>
```

Functions are called as <Activity Name>.<Method Name>. Here are examples of a flow and sub-flow calling UD functions in every place possible:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <FlowDefinition eventName="AllTypes" timeout="2147483647"
  created="2008-11-18T15:25:23.097+00:00"
  modified="2008-11-18T15:25:23.097+00:00"
  lastModifiedBy="magres" name="UDFunctions1"
  xmlns="http://www.sas.com/xml/schema/sas-svcs/rtdm-1.1">
  <Description>Subflow (of UDFunctions) that calls
   UD functions</Description>
- <StartNode name="myStartNode">
- <Transition name="toReply">
  <Rule>UDFunctions1StartTr.yes()</Rule>
  <DestinationNodeName>myReplyNode</DestinationNodeName>
  </Transition>
  </StartNode>
  <ReplyNode name="myReplyNode" />
  <UIData />
  </FlowDefinition>

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <FlowDefinition eventName="AllTypes" timeout="2147483647"
  created="2008-11-18T15:25:23.097+00:00" modified="2008-11-18T15:25:23.097+00:00"
  lastModifiedBy="magres" name="UDFunctions"
  xmlns="http://www.sas.com/xml/schema/sas-svcs/rtdm-1.1">
  <Description>Flow that calls UD functions from a variety of places</Description>
  <ProcessVariable name="PVInt1" type="Int" array="false" />
- <StartNode name="myStartNode">
- <Transition name="toAssignment">
  <Rule>UDFunctionsStartTr.yes()</Rule>
  <DestinationNodeName>myAssignmentNode</DestinationNodeName>
  </Transition>
  </StartNode>
- <AssignmentNode name="myAssignmentNode">
- <Transition name="toSubflow">
  <Rule>UDFunctionsAssignmentTr.yes()</Rule>
  <DestinationNodeName>mySubFlowCallNode</DestinationNodeName>
  </Transition>
  <Assignments expr="UDFunctionsAssignmentRHS.yes()"
    processVariableName="PVInt1" />
  </AssignmentNode>
- <SubFlowCallNode eventName="UDFunctions1" name="mySubFlowCallNode">
- <Transition name="toCode">
  <Rule>UDFunctionsSubflowTr.yes()</Rule>
  <DestinationNodeName>myCodeNode</DestinationNodeName>
  </Transition>
  </SubFlowCallNode>
- <CodeNode name="myCodeNode">
- <Transition name="toActivityCall">
  <Rule>UDFunctionsCodeTr.yes()</Rule>
  <DestinationNodeName>myActivityNode</DestinationNodeName>
  </Transition>
  <Expression>PVInt1 = UDFunctionsCodeCode.yes();</Expression>
  </CodeNode>
- <ActivityMethodCallNode activityMethodName="method1"
  activityName="UDFunctionsAsActivity" name="myActivityNode">
- <Transition name="toReply">
  <Rule>UDFunctionsActivityTr.yes()</Rule>
  <DestinationNodeName>myReplyNode</DestinationNodeName>
  </Transition>
  </ActivityMethodCallNode>
  <ReplyNode name="myReplyNode" />
  <UIData />
  </FlowDefinition>
```

The easiest way to package UD functions is to group related functions together as multiple methods in a single code activity.

Guidelines for Creating Activities

Date 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.

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.

Chapter 6 Integration

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 serialized back to the invoking client.

One-way event operations are also supported, but they 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 variables are also independent of the reply message sections of the event definition. Client applications should not rely on reply variables being returned in any particular order.

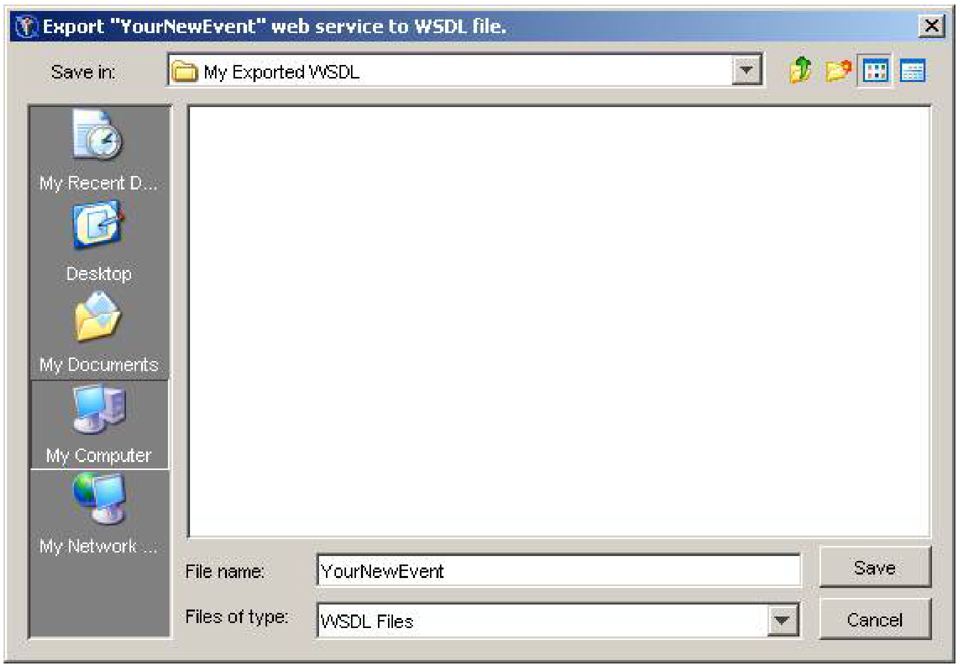
Web Service Definition Language

Retrieve a WSDL File

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

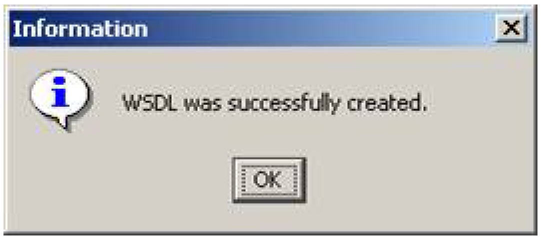
1. Open SAS Management Console.
2. On the **Folders** tab, select **System** ⇨ **Applications** ⇨ **SAS Decision Services** ⇨ **Decision Services 6.2**.
3. Navigate to the SAS Decision Services repository for which you want to generate a WSDL file.
4. Right-click the event in the repository and click **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.

Display 6.1 *WSDL File*



7. Click **Save**.

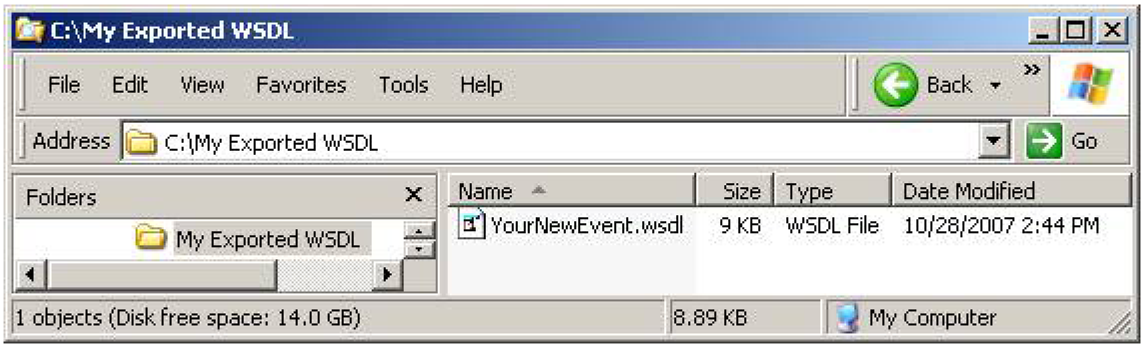
Display 6.2 *WSDL Created*


Information

WSDL was successfully created.

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

Display 6.3 *Exported WSDL*


C:\My Exported WSDL
File Edit View Favorites Tools Help
Back
Address C:\My Exported WSDL
Go
Folders
My Exported WSDL

Name	Size	Type	Date Modified
YourNewEvent.wsdl	9 KB	WSDL File	10/28/2007 2:44 PM

1 objects (Disk free space: 14.0 GB) 8.89 KB My Computer

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</ClientTimeZoneID>
                </rdm:Header>
               <rdm:Body>
                      <rdm:Data name="CustomerID>"
                            <rdm:String>
                                 <rdm:Val>001</rdm:Val>
                            </rdm:String>
                      </rdm:Data>
                      <rdm:Data name="Amount">
                            <rdm:Float>
                                 <rdm:Val>25000.0</rdm:Val>
                            </rdm:Float>
                      </rdm:Data>
                      <rdm:Data name="Mood">
                            <rdm:String>
                                 <rdm:Val>Good</rdm:Val>
                            </rdm:String>
                      </rdm:Data>
                      <rdm:Data name="SigEvent">
                            <rdm:String>
                                 <rdm:Val>NewBaby</rdm:Val>
                            </rdm:String>
                      </rdm:Data>
                </rdm:Body>
            </rdm:Event>
        </env:Body>
</env:Envelope>
```

ClientTimeZoneID is a required tag in the SAS Decision Services header. Time zone names from the public domain time zone (TZ) database are accepted. The following website lists time zone information from the TZ database: **http://home.tiscali.nl/~t876506/TZworld.html#nam**.

Every web service stack has client tools that can be used to generate both stubs and helper classes that call particular web services. These toolsets take the desired 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.

Integration with SAS Model Manager

About SAS Model Manager

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.

SAS Model Manager 12.1 is the required version for use with SAS Decision Services.

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 Decision Services 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.

Integration with SAS Data Surveyor for Clickstream Data

About SAS Data Surveyor for Clickstream Data

The term *clickstream* describes the data that is collected from users as they access websites through various electronic devices. Clickstream data includes the stream of user activity that is stored in a log. Clickstream data can be collected and stored in a variety of ways. SAS Data Surveyor for Clickstream Data enables you to process this data and produce meaningful results.

Integration of SAS Data Surveyor for Clickstream Data with SAS Decision Services allows for real-time campaign content to be presented to the web site visitor. The real-time content is based on information that is specific to the visitor's session. Any subsequent activity that the user takes on the presented content is tracked. This tracking can help with determining the success of campaigns and analyzing customers responses to different types of content that are presented within a campaign.

Real-Time Behavior Tracking and Analysis

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

See *SAS Data Surveyor for Clickstream Data User's Guide* for information about configuration and use of the combined functionality provided by SAS Data Surveyor for Clickstream Data and SAS Decision Services.

Decision Logic Smart Object

Overview

The Decision Logic smart object contains Decision Services XML artifacts, as well as generated code that is based on those artifacts, for specific target systems. The only supported target is SAS Data Integration Studio, which uses generated DS2 code. Specific details about using that code follow. The object is created through SAS Decision Manager, where a top level flow and any sub-flows are selected. SAS Decision Manager, in turn, calls the SAS Decision Services design server API and passes it the names of the flows to include. The design server creates the smart object. See SAS Decision Manager documentation for information about this process. After it is created, the Decision Logic smart object is an independent entity within the SAS Metadata Repository.

The Decision Logic smart object can be imported, exported, copied, and renamed. It can be placed in any folder in the SAS Metadata Repository. Decision Logic smart object cannot be updated. To change an existing Decision Logic smart object, you must re-create it.

Data Integration Target

Packages

DS2 packages are created for all of the Decision Services artifacts including flows, activities, and global variables. Events and resources are used as part of the code generation process but do not cause any packages to be created for themselves.

Note: If an event and an activity share the same name, and if both are included in a smart object export, DS2 code generation fails and the smart object is not created. This limitation does not apply to events and activities in the real-time engine environment.

Macro Variables

There are four macro variables that are used by the execution code that is generated for the data integration target:

ds_batch_job_id
: This variable is used to identify this run of the code. It is put into a column in the output table to identify which run the response was from. It is also used in the stats table for the same reason.

ds_input_table
: This is the source for the input rows. Use the getInputVariables() method on the smart object to determine the columns that are expected for this table. The value should be in the form of libname.tablename (or just tablename for the work library). A character column that is named correlation_id with a minimum length of 32 characters must always be present in the input table and will always be included in the values that are returned by the getInputVariables() method. The value in this column is used to match the input transaction row with the result transaction row in the output table.

ds_output_table
: This is the target for the output rows. Use the getOutputVariables() method on the smart object to determine the columns that are expected for this table. The value should be in the form of libname.tablename (or just tablename for the work library). Rows are appended to the end of the table. Character columns that are named batch_job_id (minimum length 32 characters), correlation_id (minimum length 32 chracters), and status_cd (minimum length 10 chracters) must always be present in the output table and will always be included in the values that are returned by the getOutputVariables() method. The batch_job_id column is used to identify which run these output rows are from and contains the value that is set in the macro variable ds_batch_job_id. This is necessary because data is appended to the output table. Therefore, it is possible to use the same output table for multiple batch runs. The status_cd column is the status code for the output and is used to indicate whether there was an error or whether the transaction succeeded.

ds_stats_table
: This is the target for the statistics information. Use the getStatsVariables() method on the smart object to determine the columns that are expected for this table. The value should be in the form of libname.tablename (or just tablename for the work library).

Set this value to blank to indicate that statistics should not be collected for this run of the code.

Execution Code

The data integration target has a data statement,generated for it, that executes the top level flow that is associated with the smart object. For each row in the input table, the top level flow is executed, and the response is written to the output table. When the run is completed, the statistics are written to the stats table if a table name has been provided.

Note: All output is appended to the tables and is distinguished by the batch_job_id.

Chapter 7 Installation

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 the *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 the *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 multiple 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.

DataFlux Authentication Server

The DataFlux Authentication Server is part of the DataFlux Data Management Platform. The platform provides centralized data access and data analysis for the data that is stored in DataFlux servers and databases across your enterprise.

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

For more information, see the *DataFlux 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 the *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 Drivers for Federation Server is used to add users and groups to the authentication server.

DataFlux Management Secure

If you use SAS/SECURE for your SAS servers, then you are required to purchase DataFlux Management 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 this 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. It is often impractical to attempt 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.

Component	Used By	Performance Critical?	Tier

SAS Management Console	Design and Run time	No	Client
SAS Metadata Server	Design and Run time	No	SAS
Third-party Database Management System	Design and Run time	Yes	Database (DBMS)
SAS Decision Services 6.2 Design Server	Design	No	Middle
SAS Decision Services 6.2 Engine	Run time	Yes	Middle
SAS Decision Services 6.2 Monitor	Run time	No	Middle
SAS Federation Server 3.2	Design and Run time	Yes	Compute
DataFlux Authentication Server 3.2	Design and Run time	No	Compute
DataFlux Data Management Studio 2.41	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. It can also be used for integration or performance testing. 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.
- The SAS Federation Server has approximately twice the throughput capability as the engine. Therefore, an optimized deployment, for CPU-bound processing, includes one of the following options:

  - two engine servers per SAS Federation Server
  - assigning more powerful hardware to the engines than to the 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.
- Open Metadata Supervisor, 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.

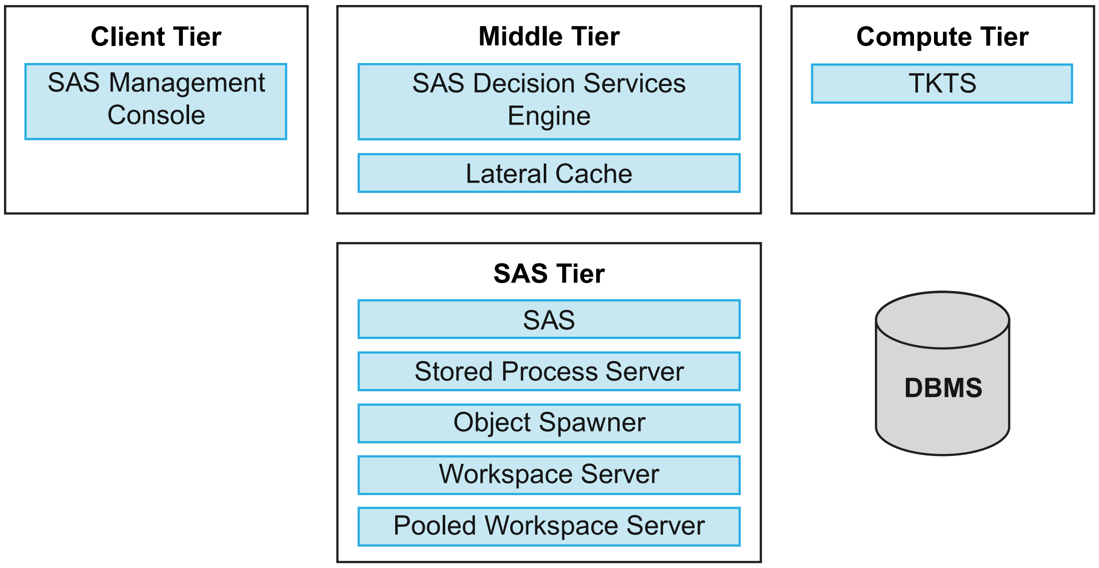
The design-time system contains a design server for creating and modifying decision services. It contains many of the same software components as a production system, in order to enable functional testing of decision services. A major difference between a design environment and a production environment 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
- one or more SAS Federation (TKTS) Server and DataFlux 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
- 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
- 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.

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 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 by far 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.

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.

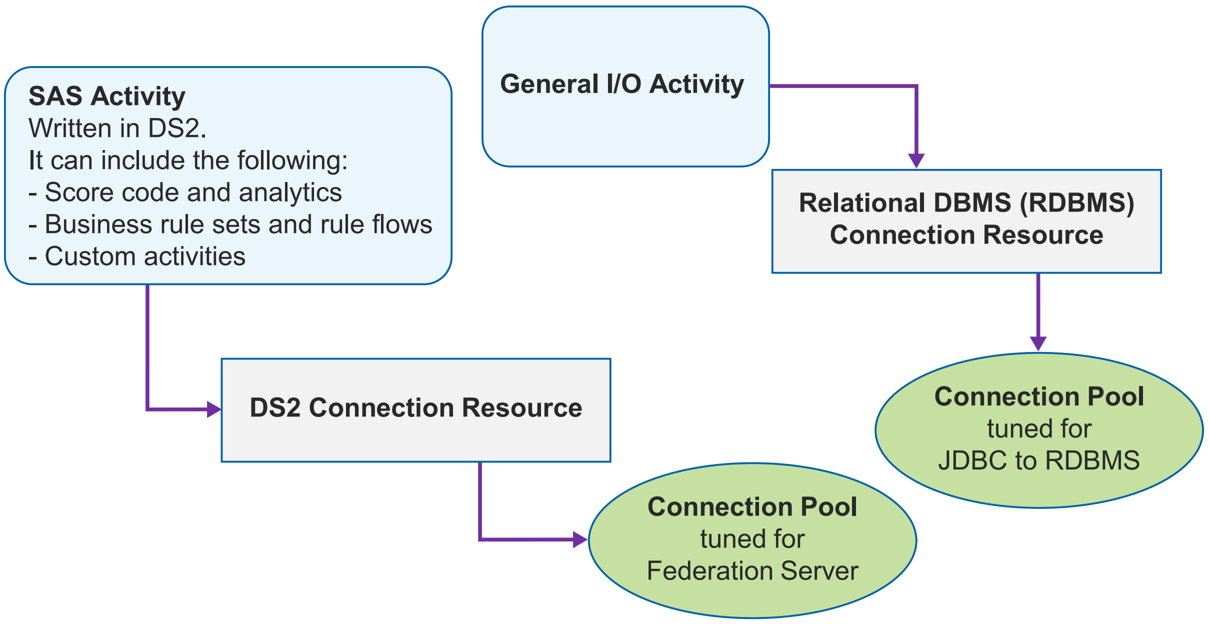
In general, processing that is highly CPU-bound performs best when the number of threads of execution in the SAS Federation Server is close to the number of cores per

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 threads from the servlet thread pool, 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*


SAS Activity
Written in DS2.
It can include the following:
- Score code and analytics
- Business rule sets and rule flows
- Custom activities
General I/O Activity
Relational DBMS (RDBMS) Connection Resource
DS2 Connection Resource
Connection Pool tuned for JDBC to RDBMS
Connection Pool tuned for Federation Server

Configuring SAS Decision Services

For more information about configuring SAS Decision Services, see the *SAS Decision Services 6.2 Configuration 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.designserver6.2.ear)
2. SAS Decision Services Engine Server (sas.decisionservices.engine6.2.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. This can assist with improving performance so long as the hardware is sufficiently powerful to run additional server instances. It can also 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.

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 6.2	(sas.decisionservices.designserver6.2.ear)
SAS Decision Services Engine Server 6.2	(sas.decisionservices.engine6.2.ear)
SAS Decision Services Monitor 6.2	(sas.decisionservices.monito6.2r.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 the *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 SAS Decision Services Monitor, using the SAS Deployment Wizard.

Chapter 8
Migration

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 DataFlux 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 DataFlux 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\
   6.2\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\
   6.2\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\
   6.2\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\
   6.2\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\
   6.2\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\
   6.2\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.

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 command 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\SASDecisionServicesMonitor6.2/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\9.2\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 this command:

   ```
   psql -h localhost -p postgres port -U
   postgres admin user DecisionServices
   ```

8. 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
   postgres admin user < dcsvc_decisionservices.sql
   ```

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

Chapter 9

Best Practices

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 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, I/O, analytics, 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.

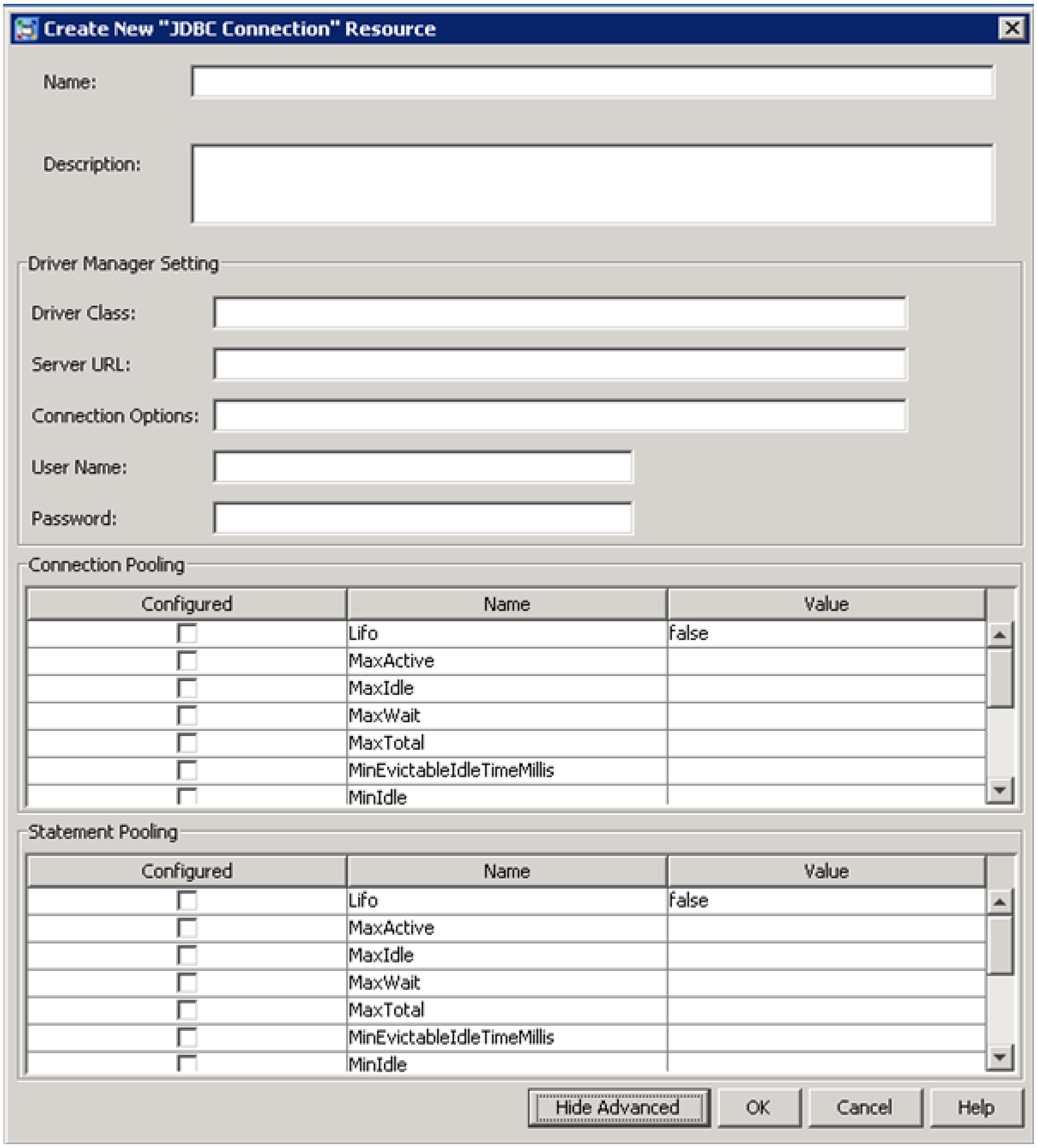
SAS Decision Services uses Apache Commons Database Connection Pooling (DBCP) to affect efficient caching and management of JDBC Connections, parameterized prepared statements, and parameterized callable statements. For more information about Apache Commons DBCP, see `http://commons.apache.org/dbcp/index.html`.

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

1. On the **Folders** tab, expand **System** ⇨ **Applications** ⇨ **SAS Decision Services** ⇨ **Decision Services 6.2**.
2. Right-click the JDBC Connection resource that you want to configure, and select **Edit System Resource**.

At a minimum, two JDBC Connection resources are needed to satisfy the different pool settings required for optimum SAS Federation Server performance versus the settings required for optimum relational database performance. Additional JDBC Connection resources can be added to enable access to additional database management systems, or to assign specific activities to specific SAS Federation Servers.

Display 9.1 *Create a JDBC System Resource - Advanced*



Name:

Description:

Driver Manager Setting

Driver Class:

Server URL:

Connection Options:

User Name:

Password:

Connection Pooling

Configured	Name	Value
☐	Lifo	false
☐	MaxActive	
☐	MaxIdle	
☐	MaxWait	
☐	MaxTotal	
☐	MinEvictableIdleTimeMillis	
☐	MinIdle	

Statement Pooling

Configured	Name	Value
☐	Lifo	false
☐	MaxActive	
☐	MaxIdle	
☐	MaxWait	
☐	MaxTotal	
☐	MinEvictableIdleTimeMillis	
☐	MinIdle	

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 be created in the JDBC resource. If the pooling control is saved with the JDBC resource, you 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.

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 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.

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 the evictor every 30 minutes and 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 SAS Federation Server connections are refreshed.

```
<TimeBetweenEvictionRunsMillis>1000 * 60 * 30</TimeBetweenEvictionRunsMillis>

<NumTestsPerEvictionRun>3</NumTestsPerEvictionRun>
```

```
<MinEvictableIdleTimeMillis>1000 * 60 * 30</MinEvictableIdleTimeMillis>
```

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 Content 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.protocl.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 the 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 49.

Web service integration

Not all of the properties that are mentioned above can be set. Also, the values of these properties are set using Java system properties and can affect all web service activities and resource combinations.

The following system properties can be set.

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.staleCheckingEnabled	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 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 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 data through the General I/O activity to preserve performance.

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 audit logging and user logging.

SAS data sets can also be used to hold DS2 activity code and score code. These are read by SAS Federation Server when a new connection is created. 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, and requires SAS Federation Server to have Read access to this shared folder.

Score code data sets are created and updated by the score code publishing stored procedure. If the SAS stored process server that executes the score code publishing code is not co-located with SAS Federation Server, the scoring data sets must be held in a shared folder and the score code publishing stored procedure requires Read or Write access to this folder.

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.

Chapter 10
Custom Configuration

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.

$Audit_Log_JDBCConnectionResource

Configured to reference a third-party database management system for the purpose of logging auditing information. Decision service activation, deactivation, and changes that were made to the values of global variables are recorded by the audit log. By default, this system resource is configured to reference the third-party database that was chosen during configuration.

$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 34. 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 113.

Table 10.1 *SAS Decision Services Checklist — Database Information*

Defaults	
Database Type:	Choose One: __DB2 9.5 or later __MS SQL 2008 or later __Oracle 10g 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 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 46.

(Optional) Create Schema Aliases

For more information about creating schema aliases, see "(Optional) Define a Schema Alias" on page 50.

(Optional) Specify Tables to Cache in Memory

For more information, see "(Optional) Specify Tables to Cache in Memory" on page 51.

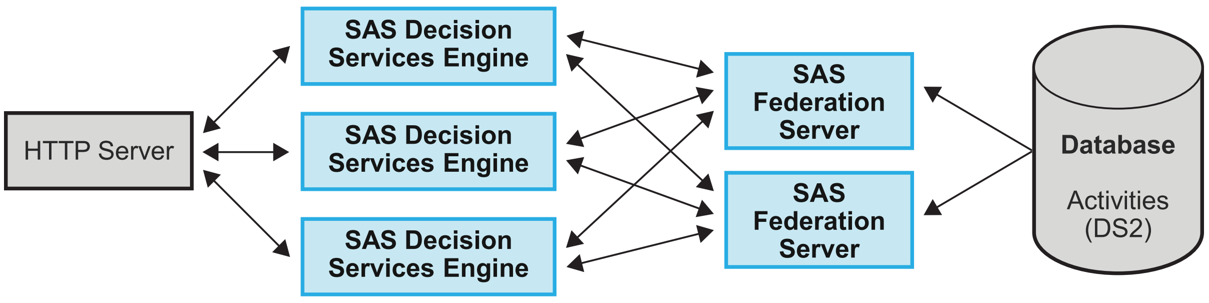
Configuring Additional SAS Federation Servers to Form a Server Cluster

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 more than one SAS Federation Server are typically configured to meet system performance and availability requirements. See Best Practices on page 103 for information about how to determine the number of servers to allocate to each tier. A minimum of two SAS Federation Servers are required to support hardware failover.

The SAS Decision Services engine attempts to load balance 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*


HTTP Server
SAS Decision Services Engine
SAS Decision Services Engine
SAS Decision Services Engine
SAS Federation Server
SAS Federation Server
Database
Activities (DS2)

Install and Configure a New SAS Federation Server

See the *SAS Decision Services 6.2 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 6.2**.
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 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 underused while the less powerful servers will 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 and audit 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, user, and audit 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\6.2\Configurable`**.
2. Copy the new JDBC JAR files into the application server lib directory.

   For JBoss, you must copy the new JDBC driver JAR files to the application server's lib directory. For WebSphere, you must configure a JDBC provider for your database. This is accomplished within the WebSphere administrator's console, where you provide the path to the JDBC driver JAR files.

   Follow the application server vendor's instructions to deploy the JDBC driver for your selected database. This procedure makes the database driver available for use by the SAS Decision Services engine.
3. Using DataFlux 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 $AuditLog, $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.

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 9.7 client
Oracle 10g client and later	Oracle Wire Protocol	Oracle 11.2 client and administrator and Oracle Enterprise Manager
TTU 12 and later (client and utilities, including TPT)	Teradata	Teradata Client 13.0
Microsoft SQL Server 3008 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 9.3 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
- DataFlux 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	<*SAS Config*> is the location for installing the configuration for SAS. Here is an example:, **`D:/SAS/Config/Lev1`** *yyyy-mm-dd* is the date time of creation of the log file. *nnnn* is a four-digit number to differentiate between logs created by multiple processes that perform the same function. *N.N* is the major and minor version number of SAS Decision Services.
Metadata Server	**`<`*`SAS Config`*`>/SASMeta/MetadataServer/Logs/SAS_Meta_MetadataServer_yyyy-mm-dd_nnnn.log`**

Component	Default Location of Logs

SAS Federation Server	`<install root>\FederationServer\3.1.1\etc\dfs_log.xml`
	`<install root>\FederationServer\3.1.1\var\log`

Middle Tier	
Remote Services	`<SAS Config>/Web/Logs/RemoteServices.log`
SAS Decision Services Engine Server	`<SAS Config>/Web/Logs/SASDecisionServicesEngineServerN.N.log`
SAS Decision Services Design Server	`<SAS Config>/Web/Logs/SASDecisionServicesDesignServerN.N.log`
SAS Decision Services Monitor	`<SAS Config>/Web/Logs/SASDecisionServicesMonitorN.N.log`
SystemOut	`<WAS Base>/profiles/<profile name>/logs/<server name>/SystemOut.log`
SystemErr	`<WAS Base>/profiles/<profile name>/logs/<server name>/SystemErr.log`
SAS Web Application Server	`<SAS-config-dir\Lev1\Web\WebAppServer\SASServer7_1\logs\server.log`

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 of 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 6.2** and select **Properties**. On the **Advanced** tab, change the policy.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.

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 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.

Parameter	In/Out	Type	Description
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	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: • 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. 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." 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 much be present or an error is returned.

Parameter	In/Out	Type	Description
statsLibrary	In	String	The library or schema containing the node-level counts table. This refers to a SAS Decision Services library resource entry.
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 135.
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.

Parameter	In/Out	Type	Description
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.
timeZoneColumnOrId	In	String	This parameter is ignored (can be null) when timeZoneOptions = NONE. When timeZoneOptions = GLOBAL, it specifies the time zone identifier to use as the client timezone for the entire simulation. When timeZoneOptions = COLUMN, it specifies the column of inTable 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	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 INITIALIZING. During the IN_PROGRESS phase, these numbers reflect the actual transactions processed and can be used to track progress. If the state is COMPLETE, the total number of transactions should be a sum of the number completed and the number that had errors. Generally, the FATAL_ERROR can be entered during the initialization phase. In this case, the counts are all zeros. If a fatal error is encountered during the processing, the state is set to FATAL_ERROR and the counts reflect the transactions that were processed when the fatal error took place.
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).
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	BatchTimeZoneOver ride	There are three options: NONE, GLOBAL, or COLUMN.

Parameter	In/Out	Type	Description
timeZoneColumnOrId	In	String	This is ignored (can be null) when timeZoneOptions = NONE. When timeZoneOptions = GLOBAL, it specifies the time zone identifier to use as the client timezone for the entire batch run. When timeZoneOptions = COLUMN, it specifies the column of inTable 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)
```

Appendix 4

Activate Flows Using BatchActivator

`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.
-password <*password*>	User login password. Required if -profile is not set or if the profile does not contain connection credentials.
-port <*port*>	Metadata server port. Required if -profile is not set.
-profile <*profile*>	Metadata server connection profile. Can be used in place of the -host, -port, -user, and -password options.
-user <*userid*>	User login identity. Required if -profile is not set or if the profile does not contain connection credentials.

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