THE POWER TO KNOW.

SAS® 9.3 Foundation Services
Administrator's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS® 9.3 Foundation Services: Administrator's Guide*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Foundation Services: Administrator's Guide

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

1st electronic book, July 2011

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

Contents

What's New in SAS 9.3 Foundation Services

Overview

In SAS 9.3 Foundation Services, the Logging service is deprecated.

Logging Service Deprecation

The Logging service is deprecated in SAS 9.3. Custom SAS clients should use Log4j to perform logging tasks.

Chapter 1

Concepts

Overview of SAS Foundation Services

SAS Foundation Services includes tools to enable application development and service administration for the SAS Foundation Services. Depending on the components that you choose to install, SAS Foundation Services includes one or more of the following components:

- SAS Management Console plug-ins, which enable you to administer configuration metadata in a metadata repository. The following plug-ins can be installed with the SAS Foundation Services:
  - Application Monitor, which enables administrators to monitor the performance and activities of a foundation service-enabled application.
  - Foundation Services Manager, which enables administrators to define and manage service deployments and service configurations.
  - Publishing Framework Manager, which enables administrators to set up metadata for users and applications to do the following:
    - publish SAS files to a variety of destinations
    - receive and process published information

  For more information about SAS Foundation Services administration, see the online Help for the appropriate administrative plug-in.
- SAS Foundation Services, which is a set of platform infrastructure and extension services for programmers who want to write applications that are integrated with the SAS platform. For information about coding applications that use the SAS Foundation Services, see Chapter 4, “Using SAS Foundation Services,” in *SAS Integration Technologies: Java Client Developer's Guide* in the *SAS Integration Technologies: Java Client Developer's Guide*.

  The following table presents the function and related documentation for each of the SAS Foundation Services:

Table 1.1 *SAS Foundation Services*

Service	Java Class	Function	Related Documentation
Connection Service	com.sas.services.connection.platform	IOM connection management	For details about administering the SAS servers that you connect to with the Connection Service, see the *SAS Intelligence Platform: Application Server Administration Guide*. For development information and coding examples, see Chapter 2, "Using the Java Connection Factory," in *SAS Integration Technologies: Java Client Developer's Guide* in the *SAS Integration Technologies: Java Client Developer's Guide*.
Discovery Service	com.sas.services.discovery	locating and binding to deployed services	For details about how applications use the Discovery Service, see "Understanding How Applications Locate Foundation Services " on page 22 .
Event Broker Service	com.sas.services.events.broker	asynchronous event notification and request management to support dynamic, event-driven processes	For details about editing the Event Broker Service configuration, see "Modifying the Session and User Service Configurations" on page 49 . For information about using the Publishing Framework to generate and publish events, see the *SAS Publishing Framework: Developer's Guide*.
Information Service	com.sas.services.information	repository federation, searching repositories, a common entity interface, and creating personal repositories	For details about editing the Information Service configuration, see "Modifying the Information Service Configuration " on page 41 .
Logging Service	com.sas.services.logging	run-time execution tracing, response metric and resource utilization reporting, and error tracking	For details about editing the logging service configuration, see "Modifying the Logging Service Configuration " on page 44 .
Publish Service	com.sas.services.publish	access to the publication framework	For details about the Publishing Framework, see the *SAS Publishing Framework: Developer's Guide*.

Service	Java Class	Function	Related Documentation
Security Service	com.sas.services.security	user authentication, propagation of user identity context across distributed security domains, and protected-resource access policy administration and enforcement	For detailed information about implementing security in your environment, see the *SAS Intelligence Platform: Security Administration Guide*.
Session Service	com.sas.services.session	context management, resource management, and context passing	For details about editing the session service configuration, see "Modifying the Session and User Service Configurations" on page 49 .
Stored Process Service	com.sas.services.storedprocess	access to stored process execution and package navigation	For details about stored processes, see the *SAS Stored Processes: Developer's Guide*.
User Service	com.sas.services.user	access to authenticated user context, access to global, solution-wide, and application-specific profiles, and access to personal objects	For details about editing the User Service configuration, see "Modifying the Session and User Service Configurations" on page 49 .

In addition, you use the deployment utilities (com.sas.services.deployment) to deploy the services.

This document covers the following SAS Foundation Services topics:

service deployments
: In order to use the foundation services in your applications, you must deploy the services. To deploy the services, you must configure a service deployment. To understand service deployments and service deployment configuration, see "Understanding Service Deployments" on page 4 and "Understanding Service Deployment Configuration" on page 7 .

service deployment definitions
: To define and manage service deployments, see the Managing Service Deployments topics.

installing and running SAS Foundation Services as a Windows service
: The Java Service Wrapper from Tanuki Software is provided with SAS Foundation Services. You can use this software to install and run SAS Foundation Services as a Windows service for use with any application that uses SAS Foundation Services. For details, see "Overview of Installing and Running Foundation Services as a Windows Service" on page 17 .

applications that are enabled for SAS Foundation Services
To understand how applications deploy, locate, and share services, see the following topics:

- "Understanding How Applications Deploy Foundation Services" on page 21
- "Understanding How Applications Locate Foundation Services " on page 22
- "Understanding How Applications Share SAS Foundation Services" on page 28

service configurations
To understand how to modify the configurations of certain foundation services, see "Overview of Modifying Service Configurations" on page 31 .

Understanding Service Deployments

Overview of Service Deployments

A service deployment is a configuration of a collection of SAS Foundation Services that specifies the data that is necessary to instantiate the services, as well as dependencies on other services. You create service deployments for applications that deploy or access the services. You can store the service deployment configuration in the following locations:

SAS Metadata Repository
You can use the Foundation Services Manager plug-in (of SAS Management Console) to administer service deployment metadata that is stored in a SAS Metadata Repository. The SAS Metadata Server controls access to the metadata.

XML file
You can export service deployment metadata from the SAS Metadata Server to an XML file. You can then use the XML file to import service deployment metadata into another SAS Metadata Repository. If you use an XML file to store service deployment metadata, then there is no administration or access control for the metadata in the XML file.

container deployment file:
You can export service deployment metadata from the SAS Metadata Repository to a container deployment file. You can then deploy the SAS Foundation Services configuration in a Spring Framework container.

Note: It is recommended that you store the service deployment metadata on a SAS Metadata Repository. Storing the service deployment metadata in a SAS Metadata Repository enables it to be updated and queried from one centralized location.

To enable your application to deploy and access the foundation services, you can create local or remote service deployments:

local service deployment
supports exclusive access to a set of services deployed within a single Java Virtual Machine (JVM). Use a local service deployment when you want your application to have its own exclusive set of foundation services.

remote service deployment
supports shared access to a set of services that are deployed within a single JVM, but are available to other JVM processes. Use a remote service deployment when you want to share a foundation service deployment among multiple applications. When you create services for remote service deployments, you must specify that the

services will be accessed remotely. In order to allow remote access to the services, you must also create a service registry and associate named services with the named components for the remote services.

A service deployment contains:

service deployment groups
: When you create service deployments (local or remote), you can also create groups within the service deployment in order to organize services within a deployment hierarchy.

services and service initialization data
: Within each service deployment group, you must define the services for that group. Service definitions contain the following information:

  service types (interfaces)
  : designate which service interfaces are implemented by the service. The Discovery Service is locates services according to their service interfaces. For example, if you want to locate a service that implements a Logging Service interface, have the Discovery Service search for a service that implements **`com.sas.services.logging.LoggingServiceInterface`**.

    Note: All SAS Foundation Services (including local services) implement the Remote Service interface.

  service configuration
  : specifies the Java class that is used to create the service, the service's optional configuration data, and the service's configuration user interface. The service configuration user interface defines the Java class used by the Foundation Services Manager to configure the service's configuration details.

  service dependencies
  : specify other services on which the service depends. When they are deployed, foundation services might depend on the availability of one or more other foundation services. When you define a service, you must specify the other services on which that service depends. For example, the Authentication Service uses the Logging Service. Therefore, when you define the Authentication Service in a service deployment, you must specify the Logging Service as a dependency.

  service names (for remote access only)
  : specify named services for remote access. if a foundation service is to be made available for remote clients, you must enable the service for remote access and define named services (service names) that specify the service's name bindings to one or more service registries.

  consumers (application configurations)
  : group the resources that are associated with a specific application. If several applications share a single service configuration, then you can group the resources in your service configurations by defining consumers. Consumers enable you to install and uninstall the specific parts of a service configuration that are associated with a specific application. For example, if your Logging Service configuration contains a renderer that is specific to SAS Web Report Studio and you uninstall SAS Web Report Studio, then the consumer definition can be used to remove that specific renderer while leaving the rest of the Logging Service configuration intact.

  authorization permissions
  : enable you to specify which user or group identities can perform which actions on a particular resource.

Note: If a service depends on other services, then you must define those services before defining the service that depends on them. For details about service dependencies and order of definition, see "Service Dependencies" on page 6 .

service registries and associated named services (remote service deployments only)
To enable services for remote access, you must define a service registry to use in locating remote services. (A service registry is a searchable registry of service descriptions that is used to register named service bindings). You must then register the services with the service registry by creating or associating named services that define how each service is to be used within the context of the Discovery Service.

To understand where service deployments are defined, see "Understanding Service Deployment Configuration" on page 7 .

Service Dependencies

If a service has a dependency on another service, you must first create the service on which it depends. The following table shows the service dependencies and the relative order in which you must define the services.

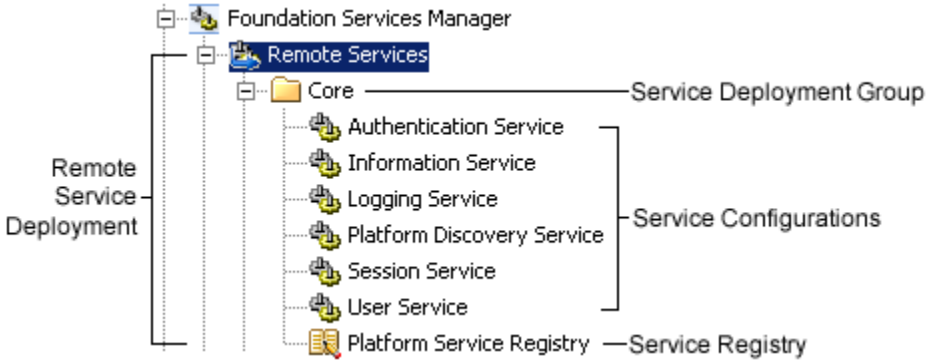
Table 1.2 *Service Dependencies Table*

Service	Service Dependencies
Logging Service	
Authentication Service	
Discovery Service	
Information Service	Logging Service
User Service	Logging Service Authentication Service Information Service
Session Service	Authentication Service Logging Service Information Service User Service
Event Broker Service	Logging Service Authentication Service Information Service User Service Session Service
Stored Process Service	Logging Service

Understanding Service Deployment Configuration

The following figure shows the SAS Management Console Foundation Services Manager connected to a SAS Metadata Repository that contains a service deployment named Remote Services. The figure also points to the service deployment group, services, and service registry defined within the Remote Services service deployment.

Figure 1.1 *SAS Foundation Services Deployment*



You can define a service deployment in a SAS Metadata Repository in one of the following ways:

- Use the Foundation Services Manager Plug-in of SAS Management Console to create a service deployment. For details, see "Defining Service Deployments" on page 9 .
- Import an XML file that contains the service deployment. If your application's service deployment configuration is contained in an XML file, you can import it into a SAS Metadata Repository. For details, see "Importing Service Deployments" on page 12 .

After you import or create a service deployment, you can do the following:

- Export the service deployment to an XML file. If an application does not have access to a SAS Metadata Repository in its run-time environment, you can export the service deployment configuration to an XML file that the application can access for service deployment configuration information. For details, see "Exporting Service Deployments" on page 13 .
- Export the service deployment to a container deployment file. You can export the service deployment configuration to a container deployment file that can be deployed in a Spring Framework container. For details, see "Exporting Service Deployments" on page 13 .
- Duplicate the service deployment. If you need to use a service deployment that is similar to an existing service deployment, you can duplicate an existing service deployment configuration. For details, see "Duplicating Service Deployments" on page 14 .

In addition, you might need to update the prototypes that define the foundation services. (To update prototypes, select the Foundation Services Manager and select **Actions** ⇨ **Update Prototypes**). For further information about using the Foundation Services Manager to create service deployments, see the Foundation Services Manager Help.

Chapter 2

Managing Service Deployments

Defining Service Deployments

Overview of Defining Service Deployments

You create service deployments for applications to deploy and access SAS Foundation Services. Applications deploy service deployments by using the service deployment name that is configured in a SAS Metadata Repository or XML file. To understand the components of service deployments, see "Understanding Service Deployments" on page 4 .

To create a service deployment:

1. Create a service deployment.
2. Create service deployment groups for your service deployment.
3. Create services within each service deployment group.
4. For services that can be accessed remotely, create a service registry and associated named services.

Step 1: Create a Service Deployment

To create a service deployment by using the Foundation Services Manager, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.

2. In the navigation tree, select **Foundation Services Manager** and then select **Actions** ⇨ **New Service Deployment** from the main menu. The New Service Deployment window appears.
3. Enter a name and an optional description, and then click **Next**.
4. (Optional) Specify name and value pairs that can be used to look up services.
5. Click **Finish** to define the service deployment.

Step 2: Create Service Deployment Groups

To create service deployment groups, perform the following steps:

1. In the SAS Management Console navigation tree, select the service deployment in which you want to create a new service deployment group. Select **Actions** ⇨ **New Service Development Group**. The New Service Deployment Group window appears.
2. Enter a name and an optional description, and then click **Next**.
3. (Optional) Specify name and value pairs that can be used to look up services. Click **Next**.
4. Click **Finish** to create the new service deployment group.

After you create the deployment group, you can select the deployment group and do one of the following:

- For local service deployments, create new services within that service deployment group
- For remote service deployments, create the services registry within that service deployment group.

Step 3: Create a Service

If a service is dependent upon other services, you must define those services before defining the service that depends on them. For details about service dependencies and order of definition, see "Service Dependencies" on page 6 .

To create a new service, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, locate and select the service deployment group where you want to create a new service. Select **Actions** ⇨ **New Service**. The New Service wizard appears.
3. Select a service to use as a prototype. Click **Next**.
4. Enter a name and an optional description, and then click **Next**.
5. The service interfaces that are associated with the selected service are displayed. Click **Next**.
6. If the service has customizable configuration data, then click **Configuration** to supply the configuration information. For details, see "Overview of Modifying Service Configurations" on page 31 .

   When you have finished editing the service configuration, click **Next**.

7. (Optional) To make this service a remote service, select the **Enable remote clients to access service capabilities** check box and then click **Service Names**. The Service Names window appears.

   To create a named service, perform the following steps:

   a. Click **New** to create a new named service.

   b. Enter the name and an optional description. Click **Next**.

   c. In the New Named Service wizard, enter the name of the binding and then select the type of binding (bind or rebind). If you are creating an Event Broker Service, then enter a codebase. If you are creating a new named service for a service registry, click **Select** to select the named component that is associated with this named service.

      If you want to associate the named service with a service registry, then move that service registry from the **Available** pane to the **Used** pane.

      Click **Next**.

   d. Review the named service definition, and then click **Finish** to save the named service definition and return to the Service Names window.

   e. When you have finished creating named services, click **OK**.

   f. Click **Next**.

8. Select the services that your new service requires. Click **Next**.

9. (Optional) Specify name and value pairs that can be used to look up services. Click **Next**.

10. Select the services that your new service requires by moving the services from the **Available Services** pane to the **Required Services** pane. For more information about service dependencies, see "Service Dependencies" on page 6 .

    Click **Next**.

11. If you are creating a new Event Broker Service, perform the following steps:

    a. Enter the default event name for the Event Broker Service. Click **Next**.

    b. Specify the resources for the Event Broker Service. Click **Next**.

    c. Specify the administrator port and transport monitors for the Event Broker Service. Click **Next**.

12. Review the service definition, and then click **Finish** to save the new service.

You can create additional services for your service deployment. If the service is enabled for remote access, you must create a new service registry.

Step 4: Create a Service Registry and Named Services

To create a service registry and named services, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.

2. In the navigation tree, locate and select the service deployment group where you want to create a new service registry. Select **Actions** ⇨ **New Service Registry** from the main menu. The New Service Registry wizard appears.

3. Select the type of service registry that you want to define. Click **Next**. The New Service Registry wizard appears.
4. Enter a name and an optional description, and then click **Next**.
5. The service interfaces that are satisfied by this definition of a service registry are displayed. Click **Next**.
6. Specify the host name and port number that is used to bind to the service registry. Click **Next**.
7. If you have not already defined the appropriate remote accessible service, then perform the following steps to define named services:
   a. Click **New** to define a new named service. The New Named Service wizard appears.
   b. Enter a name and an optional description, and then click **Next**.
   c. Enter the name of the binding and select the type of binding (bind or rebind). If you are creating an Event Broker Service, then enter a codebase. Click **Select** to select the named component that is associated with this named service. Click **Next**.
   d. Review the named service definition, and then click **Finish** to save the named service and return to the New Service Registry wizard.

   When you are finished creating new named services, click **Next**.
8. Review the service registry definition, and then click **Finish** to create the service registry.

After you create the service registry, you can select the service deployment group for the registry and create the services, including the named services associated with the service registry.

Importing Service Deployments

The Foundation Services Manager enables you to import an XML file that contains the metadata necessary to create a service deployment in the Foundation Services Manager. To import a service deployment, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, select **Foundation Services Manager**, and then select **Actions ⇨ Import Service Development** from the main menu.
3. A file selector window appears. Select the deployment file that you want to import, and then click **Open**.
4. The Import Service Deployment window appears. Specify the name for the service deployment, and then click **OK**.

Exporting Service Deployments

You can export the metadata for a service deployment to an external file. You can select either of the following file types:

service deployment file
: an XML file that contains foundation services metadata. You can use this file to import your deployment into SAS Management Console, or you can use the service deployment metadata directly from the XML.

container deployment file
: an XML file that enables you to deploy SAS Foundation Services in a Spring Framework Container.

To export a service deployment XML file, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, select the service deployment that you want to export. Select **Actions** ⇨ **Export Service Deployment to** ⇨ **Service Deployment File** from the main menu. The Export Service Deployment window appears.
3. Select the service deployment that you want to export in the **Application Service Deployments** field. Specify an output file in the **Export File** field.
4. Click **OK** to export the file and close the window.

To export a container deployment file, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, select the service deployment that you want to export. Select **Actions** ⇨ **Export Service Deployment to** ⇨ **Container Deployment File** from the main menu. The Create Service Deployment wizard appears.
3. Select the container type, and then click **Next**.
4. Specify whether you want to select specific deployment groups. Click **Next**.
5. If you specified that you want to select specific deployment groups, then select the deployment groups that you want to include. Click **Next**.
6. Select whether the foundation services metadata is stored in the SAS Metadata Repository or in an XML file (**URL**). Click **Next**.
7. Depending on where the metadata is stored, specify one of the following:
   - If your metadata is stored in the SAS Metadata Repository, then specify the connection information for the server. Click **Next**.
   - If your metadata is stored in an XML file, then specify the location where the XML file is located, and then click **Next**.
8. Review your configuration information in the **Properties** panel, and then specify the filename for the container deployment file in the **File** field. Click **Finish** to create the container deployment file.

Duplicating Service Deployments

The Foundation Services Manager plug-in of the SAS Management Console enables you to duplicate an existing service deployment under a new name. To duplicate a service deployment, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, select the service deployment that you want to duplicate. Select **Actions** ⇨ **Duplicate Service Deployment** from the main menu. The Duplicate Foundation Service Deployment window appears.
3. The **Name** field contains the name of the service deployment that you are duplicating. You must change this name to a unique deployment name. Enter a new description in the **Description** field, if needed.
4. Click **OK** to duplicate the service deployment. The new deployment appears in the navigation tree under the Foundation Services Manager.

Redistributing Service Deployments

After you have configured a remote service deployment, you might need to move your remote service deployment or service registry to a different machine.

Note: If the service deployment exists in an XML file instead of on the SAS Metadata Repository, then you must first import the service deployment before you can redistribute service deployments or service registries. See "Importing Service Deployments" on page 12 .

You can use SAS Management Console to reconfigure parameters as follows:

- move the remote service deployment to another machine. To move a remote service deployment to another machine, you must use the Foundation Services Manager to reconfigure any machine-specific service configuration data. For example, the logging service might be configured to send its output to a file in the directory **`c:\original\log.txt`** on a Windows machine. If you move the remote service deployment to a UNIX machine, you must edit the logging service configuration and change the log file directory to **`/newmachine/log.txt`**.

  Note: If the application that deploys the remote services is starting the service registry, the service registry must be located on the same machine as the remote services deployment.

- move the service registry to another machine. To move a service registry to another machine, perform the following steps:

  - Reconfigure the Service Registry definition in the service deployment. To reconfigure the service registry, use the Foundation Services Manager to update the service registry's host name and port number.

    Note: If the service registry's host name is configured as **`localhost`**, you do not need to update the configuration when you move the service registry to a different machine.

Note: You must ensure that the port that is configured for the service registry does not conflict with a port that is already in use on the new machine.

- For Event Broker Service definitions only, reconfigure any codebase property changes in the Named Services definition. To reconfigure the codebase properties, use the Foundation Services Manager to update the named service definitions on the service registry or in the service definition.
- Ensure that the application that starts the service registry is coded to call the correct host name. For details, see the SAS Foundation Services class documentation for the Deployment Service at **`http://support.sas.com/rnd/javadoc/93`**.

After you have finished using SAS Management Console to re-configure the service deployment, if the service deployment was imported into the SAS Management Console from an XML file, use SAS Management Console to export the service deployment back to an XML file. You must export or copy the file to the location where the application accesses the XML file.

Chapter 3

Installing and Running Foundation Services as a Windows Service

Overview of Installing and Running Foundation Services as a Windows Service

The Java Services Wrapper from Tanuki Software (**`wrapper.tanukisoftware.org`**) is provided with SAS Foundation Services. You can use this software to install and run SAS Foundation Services as a Windows service for use with any foundation services-enabled application. The Java Service Wrapper handles user logouts, and it also provides automatic restarts when they are required.

Note: The SAS Web Infrastructure Platform includes a separate implementation of the Java Service Wrapper that enables you to easily install the SAS Services application (which is provided with the SAS Web Infrastructure Platform) as a Windows service. For more information about using this implementation, see the *SAS Intelligence Platform: Web Application Administration Guide*.

Configuring the Java Service Wrapper

Overview of the Java Service Wrapper

When you install SAS Foundation Services, a master **`wrapper.conf`** file is created in your SASHOME directory. Additional application-specific **`wrapper.conf`** files are created for any programs that use the Java Service Wrapper.

In most cases, you can use the default **`wrapper.conf`** files without making any changes. Instead, you can use command-line arguments in the executable scripts to

override the configuration settings. The **wrapper.conf** file contains the following configuration directives:

wrapper.java.command
specifies the fully qualified path to java.exe.

wrapper.java.mainclass
specifies the name of the class that will be instantiated by wrapper.exe. This class must contain a main method and must implement WrapperListener. In the sample configuration file wrapper.conf, this directive specifies a class called **com.sas.services.deployment.servicewrapper.ServiceWrapperImp 1**, which is provided with SAS Foundation Services. When this class is instantiated by wrapper.exe, it registers itself as an event listener and takes action whenever the native wrapper signals an event. For more information, see the class documentation at **http://support.sas.com/rnd/javadoc/93**.

wrapper.java.classpath1
specifies the location of the java archives that you want to load.

wrapper.app.parameter.1
specifies the metadata source configuration file, which specifies the location of the deployment configuration for remote foundation services.

wrapper.java.library.path
specifies the location of the wrapper.dll or wrapper.so file.

wrapper.java.additional.1
specifies additional java parameters.

Setting a Dependency for the Metadata Server Service

If your deployment metadata is stored in a SAS Metadata Repository, and the SAS Metadata Server has been installed as a service on the same machine as the SAS Services application, then you need to specify a service dependency to ensure that the services start in the correct order. You can specify the service dependency by adding the following line to **wrapper.conf**:

```
wrapper.ntservice.dependency.1=Metadata-Service-Name
```

Changing Time-out Intervals

If the Java Service Wrapper is timing out while starting up or shutting down, it might be necessary to increase the time-out intervals from the default values. Add the following parameters to **wrapper.conf** as appropriate.

wrapper.startup.timeout
specifies the start-up time-out interval in seconds. The default value is 30.

wrapper.shutdown.timeout
specifies the shutdown time-out interval in seconds. The default value is 30.

Executing the Java Service Wrapper

To use the Java Service Wrapper to install and run SAS Foundation Services as a Windows service, execute the following scripts. In these scripts, you can use command-

line arguments to override any of the configuration directives that are contained in the **`wrapper.conf`** configuration file.

InstallRemoteServices
: installs SAS Foundation Services as a Windows service.

UninstallRemoteServices
: uninstalls the Windows service after it has been installed.

StartRemoteServices
: executes SAS Foundation Services as a Java Service Wrapper console application.

Chapter 4

Understanding How Applications Interact with Foundation Services

Understanding How Applications Deploy Foundation Services

Applications can access service deployments from a SAS Metadata Repository or an XML file (that contains exported metadata). Applications deploy services as follows:

- For a local service deployment, the application uses a service loader utility to instantiate and initialize the SAS Foundation Services for a local service deployment and to register the deployed services with a local Discovery Service. The application then has exclusive access to these locally deployed services. For a stand-alone deployment, you do not need to configure a Discovery Service.
- For a remote service deployment that is shared between applications, one of the applications must deploy the remote service deployment. The application uses a service loader utility to instantiate and initialize the foundation services for a remote service deployment and to register the deployed services with a local Discovery Service. The application then has local access to the services. To enable the services for remote access, the remote service deployment specifies a remote Discovery Service that registers with the service registry. The remote service deployment also contains a distributable configuration for any service that remote clients access. These remote services are registered with a remote Discovery Service. Other applications can then use the remote Discovery Service to access the remote services.

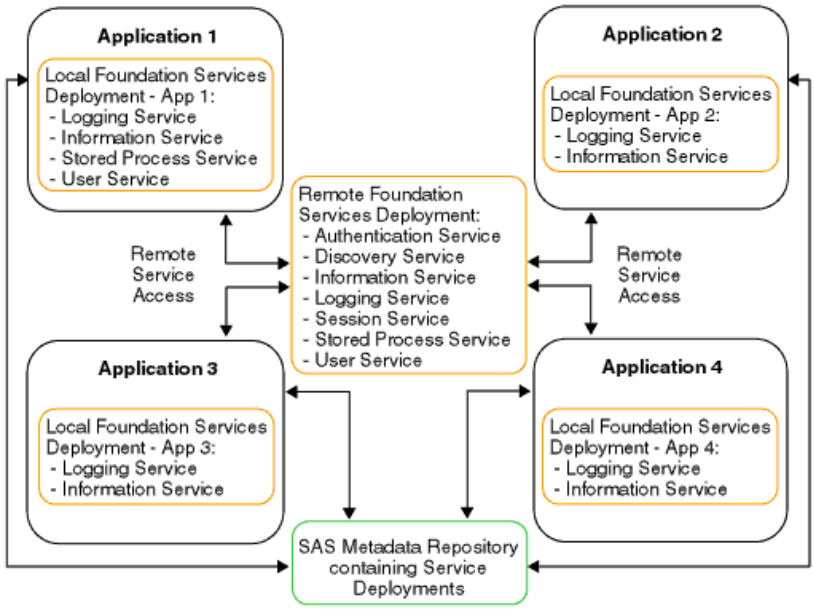
Note: A foundation service-enabled application can be either a standard client application or a Web client application that runs in a servlet container.

Your application must install the appropriate JAR files (for example, sas.svc.core.jar) in a location that is accessible only to its own classloader. This installation restriction is due to the inheritance hierarchy of classloaders. This inheritance hierarchy enables multiple applications to access classes that are available to higher level class loaders. Therefore, each foundation service-enabled application should not install the required JAR files in a

location that is accessible to a class loader that might be shared among multiple applications. For details about coding client applications for service deployment, see the SAS Foundation Services class documentation for **`com.sas.services.deployment`** and **`com.sas.services.discovery`** at **`http://support.sas.com/rnd/javadoc/93`** and the *SAS Integration Technologies: Java Client Developer's Guide*.

The following figure shows these components and how they work together.

Figure 4.1 *A Remote Service Configuration*



In the figure, Applications 1 through 4 all access their local and remote service deployment configurations from a SAS Metadata Repository.

If Application 1 deploys the remote service deployment, the services are registered with a local Discovery Service and a remote Discovery Service. Applications 2, 3, and 4 can then use the remote Discovery Service to locate and access the deployed remote services. All of the applications share the same remote service deployment. In addition, each application has exclusive access to its own local service deployment. For information about how applications locate and access services, see “Understanding How Applications Locate Foundation Services ” on page 22 .

The different components in the preceding figure might exist on the same Web server or on different Web servers. You can install your applications and deploy your services on separate machines as required by the needs of your implementation. For information about distributing service deployments, see “Redistributing Service Deployments” on page 14 .

Understanding How Applications Locate Foundation Services

Overview of How Applications Locate Foundation Services

Applications can access services that are deployed locally or remotely.

Note: Your application can be either a standard client application or a Web client application that runs in a servlet container.

To locate local and remote services, perform the following steps:

1. The application uses a service loader to instantiate and initialize local services, including its local Discovery Service.
2. The application initializes and registers the local Discovery Service with a remote Discovery Service. The application locates the remote Discovery Service by obtaining the Remote Method Invocation (RMI) registry location from a SAS Metadata Repository (or XML file that contains exported metadata) and performing an RMI name lookup on the remote Discovery Service. The remote Discovery Service enables the client to locate remotely deployed SAS Foundation Services.
3. When the application requests a service, its local Discovery Service first checks whether the service is a locally registered service.
    - If the requested service is a locally registered service, then the application binds to the local service.
    - If the requested service is not a locally registered service, then the local Discovery Service uses the remote Discovery Service to search the remote services deployment for the requested service.
        - If the requested service is not registered with the remote Discovery Service, an error is returned.
        - If the requested service is registered with the remote Discovery Service, a stub to the remote service is returned and the application can then use the remote service.

For example, in Figure 4.1 on page 22 , if an application requests the Logging Service, the application binds to the local Logging Service. If an application requests the Session Service, the application uses the remote Discovery Service to locate and bind to the remote Session Service.

Note: If the application that deploys the remote services also starts the service registry, then the service registry must exist on the same machine as that application.

The following scenarios show examples of local and remote service deployment and access.

- “Scenario: Stand-alone Application” on page 23
- “Scenario: Services That Can Be Accessed Remotely” on page 25
- “Scenario: Local and Remote-accessible Services” on page 27

Scenario: Stand-alone Application

A stand-alone application deploys services locally, uses the services, and terminates the services when they are no longer needed. If an application does not need to interact with any other applications, then it can be a stand-alone application with its own exclusive local service deployment. Services locally deployed by this application are not available to any other application. In addition, no remote services are available.

Note: An application can be either a standard client application or a Web client application that runs in a servlet container.

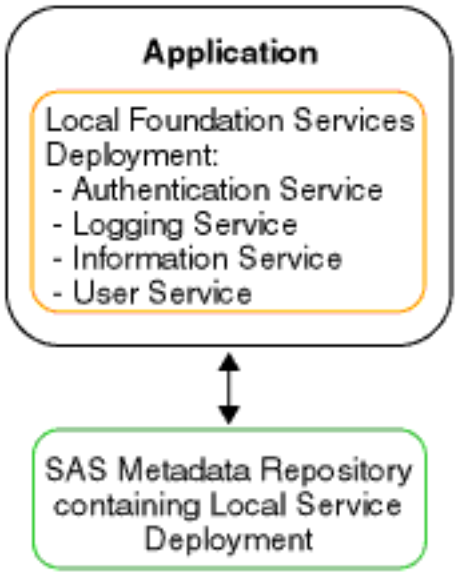
To deploy local services for its own exclusive use, the application does the following:

1. uses the service loader to query service deployment metadata from either a SAS Metadata Server or XML file (that contains exported metadata)
2. uses the service loader to instantiate services that are defined in the service deployment metadata and registers them with the local Discovery Service
3. uses the local Discovery Service to find services according to their service interfaces and optional service attributes

When the application no longer needs the services or is ready to exit, it terminates the local Discovery Service. The local Discovery Service then destroys all locally instantiated services.

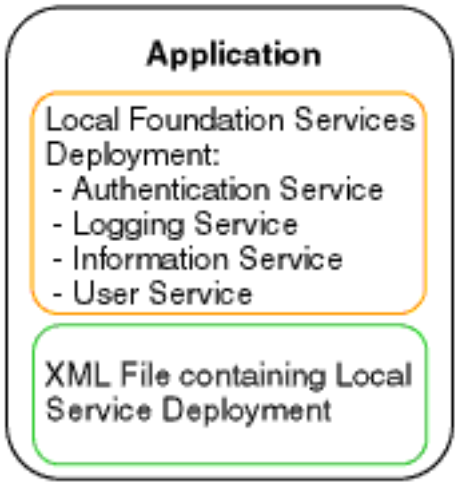
The following figure shows a stand-alone application that accesses a service deployment from a local SAS Metadata Repository:

Figure 4.2 *Stand-alone Application Accessing Local Deployment from a SAS Metadata Repository*



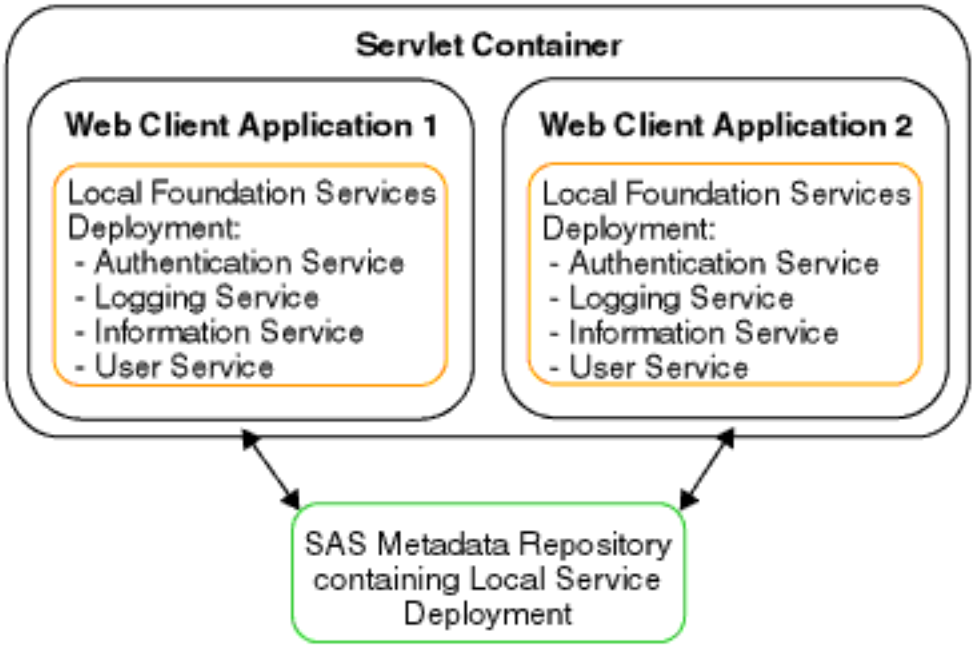
The following figure shows a stand-alone application that accesses a service deployment from an XML file:

Figure 4.3 *Stand-alone Application Accessing Local Deployment from an XML File*



The following figure shows two stand-alone Web applications that access their service deployments from a SAS Metadata Repository and each deploy their own local services for their own exclusive use:

Figure 4.4 *Two Stand-alone Web Applications Accessing Local Deployments from a SAS Metadata Repository*



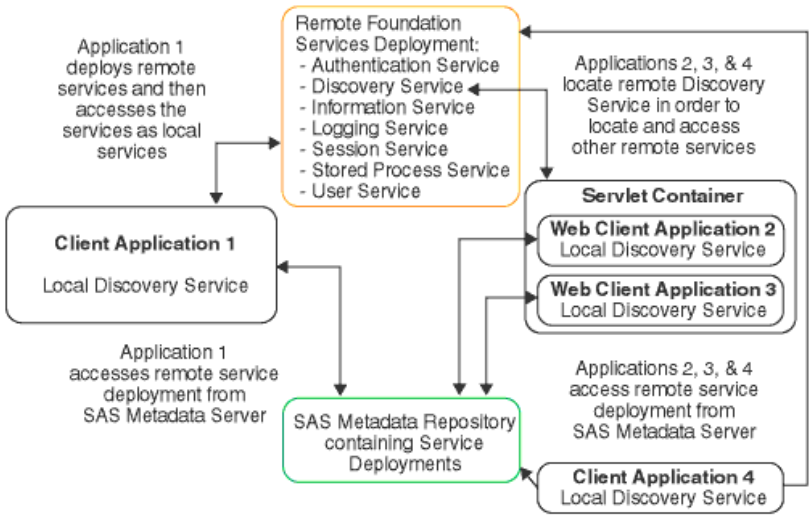
Scenario: Services That Can Be Accessed Remotely

To enable applications to access remote services, one application must deploy the remote services. (The application that deploys the remote services can then access the services as local services). Instead of deploying their own set of local services, other applications can access the remote services. To access the remote service deployment, applications locate the deploying application's remote Discovery Service in order to locate and access

the deployed remote services. This scenario is useful if one or more client applications need to use the same set of services.

In this scenario, Application 1 deploys the remote services and accesses them as local services. Applications 2, 3, and 4 locate Application 1's remote Discovery Service in order to access the remote services. Note that Applications 2 and 3 are Web client applications that run in the same servlet container and each deploy their own local services for their own exclusive use.

Figure 4.5 *Applications Accessing SAS Foundation Services Remotely*



To deploy remote services, Application 1 does the following:

1. uses the service loader to query service deployment metadata from either a SAS Metadata Server or an XML file (that contains exported metadata).
2. uses the service loader to instantiate services that are defined in the service deployment metadata and to register them with the local Discovery Service.

   Note: In this scenario, these services must be configured as accessible to remote applications.
3. uses its local Discovery Service to find services according to their service interfaces and optional service attributes.

To locate the remote-accessible services (that were deployed by Application 1), Applications 2, 3, and 4 do the following:

1. Use the service loader to query service deployment metadata from either a SAS Metadata Server or an XML file (that contains exported metadata).
2. Use the service loader to obtain a name binding to the remote-accessible Discovery Service instantiated by Application 1.
3. Register the remote Discovery Service with their own local Discovery Service.
4. Use their own local Discovery Service to find services according to their service interfaces and optional service attributes. The local Discovery Service uses the remote Discovery Service to locate the remote-accessible services.

   Note: In this scenario, Applications 2, 3 and 4 do not deploy any services themselves. They locate only services that are instantiated by Application 1.

5. When Applications 2, 3, and 4 no longer need the services, they each terminate their own local Discovery Service.

When Application 1 exits, it terminates its local Discovery Service. The local Discovery Service then terminates all locally instantiated services. After all services are terminated, no services are available to any other applications.

Scenario: Local and Remote-accessible Services

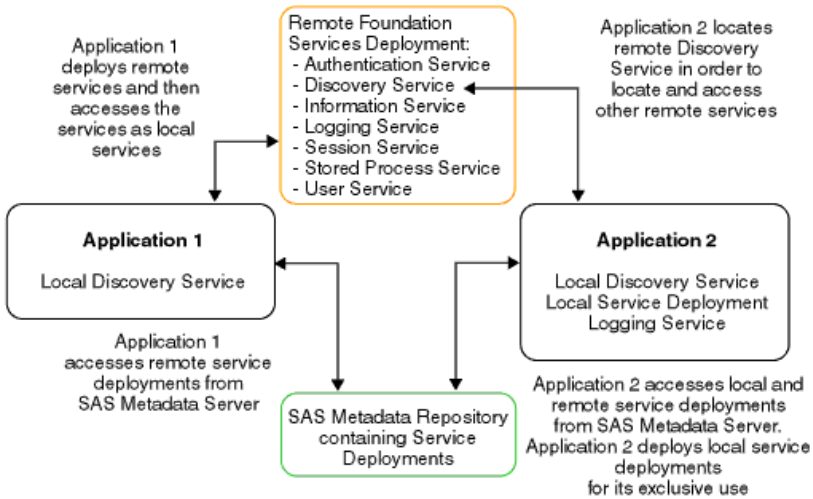
To enable other applications to access remote services, one application must deploy the remote services. (The application that deploys the remote services can then access the services as local services). Instead of deploying their own set of local services, other applications can access the remote service deployment. To access the remote service deployment, applications locate the deploying application's remote Discovery Service in order to locate and access the deployed remote services. In addition, these applications can each have their own set of locally deployed services to which each application has its own exclusive access. This example is useful when client applications need to have both of the following:

- services deployed locally for exclusive use
- use of the same set of remote services

Note: A foundation service-enabled application can be either a standard client application or a Web client application that runs in a servlet container.

In this scenario, Application 1 deploys the remote services and accesses them as local services. Application 2 locates Application 1's remote Discovery Service in order to access the remote services. Application 2 also deploys local services for its own exclusive use.

Figure 4.6 *A Scenario in Which Applications Access Both Local and Remote Service Deployments*



To deploy remote services and access these services locally, Application 1 does the following:

1. uses the service loader to query service deployment metadata from either a SAS Metadata Server or an XML file (that contains exported metadata).
2. uses the service loader to instantiate services that are defined in the metadata and to register them with the local Discovery Service.

Note: These services must be configured for remote access.

3. uses its local Discovery Service to find services according to their service interfaces and optional service attributes.

To deploy local services and access remote services, Application 2 does the following:

1. uses the service loader to query service deployment metadata from either a SAS Metadata Server or an XML file (that contains exported metadata).
2. uses the service loader to instantiate services that are defined in the metadata and to register them with the local Discovery Service.

   Note: Because these services are used only by Application 2, they are not configured for remote access.
3. uses the service loader to query service deployment metadata from either a SAS Metadata Server or an XML file (that contains exported metadata).
4. uses the service loader to obtain a binding to the remote Discovery Service that is instantiated by Application 1.
5. uses its local Discovery Service to find services based on their service interfaces and optional service attributes.

   Note: Application 2 has access to both local services and remote services. When services are located, the local Discovery Service first tries to find a service locally before it looks for a remote-accessible service.
6. When Application 2 no longer needs the services, it terminates its local Discovery Service. This causes its locally instantiated services to be destroyed and its bindings to Application 1's remote services to be terminated.

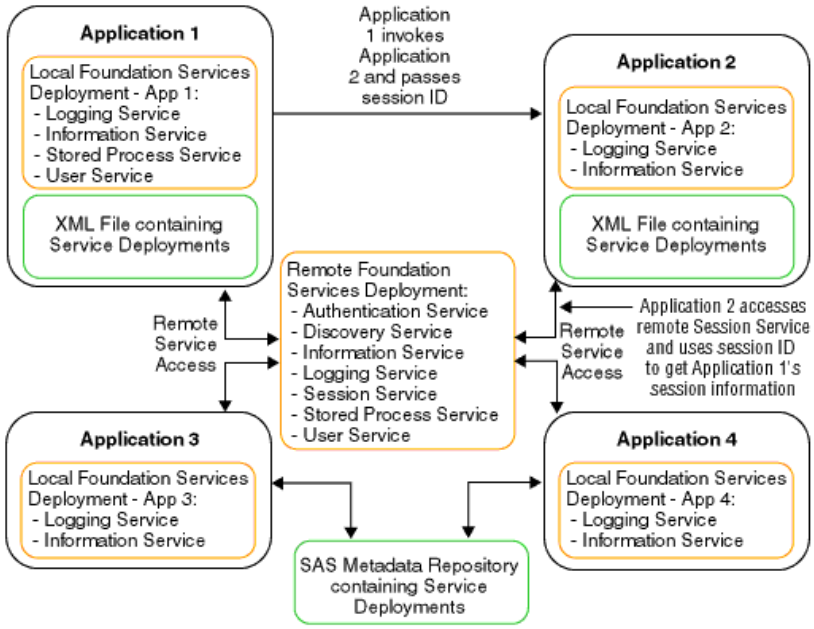
When Application 1 exits, it terminates the local Discovery Service. The local Discovery Service then terminates all locally instantiated services. After all services are terminated, no services are available to any applications.

Understanding How Applications Share SAS Foundation Services

An application can use the SAS Foundation Services to access another application's session context.

Note: A foundation service-enabled application can be either a standard client application or a Web client application that runs in a servlet container.

Figure 4.7 *Shared Services*



In the preceding figure, Applications 1 through 4 use the same remotely deployed Session Service. When Application 1 launches Application 2, it passes its session ID to Application 2. Application 2 can then bind to the remote Session Service and obtain and use Application 1's session and user context information. This allows the user to seamlessly pass through to Application 2 without requiring a separate login definition.

Chapter 5

Modifying Service Configurations

Overview of Modifying Service Configurations

After you define a service deployment and its associated services, you might want to edit the configuration information for particular services. You can use the Foundation Services Manager to modify the configuration data for the following services:

- Event Broker Service. For details, see “Understanding Events and Process Flows” on page 35 and “Modifying an Event Broker Service Configuration” on page 39 .
- Information Service. For details, see “Modifying the Information Service Configuration ” on page 41 .

- Logging Service. For details, see "Modifying the Logging Service Configuration " on page 44 .
- Session Service. For details, see "Modifying the Session and User Service Configurations" on page 49 .
- User Service. For details, see "Modifying the Session and User Service Configurations" on page 49 .

Understanding the Event Broker Service

Overview of the Event Broker Service

The Event Broker Service enables you to receive external event notifications and process them based on the name of the event that is received. Events can be structured or unstructured as follows:

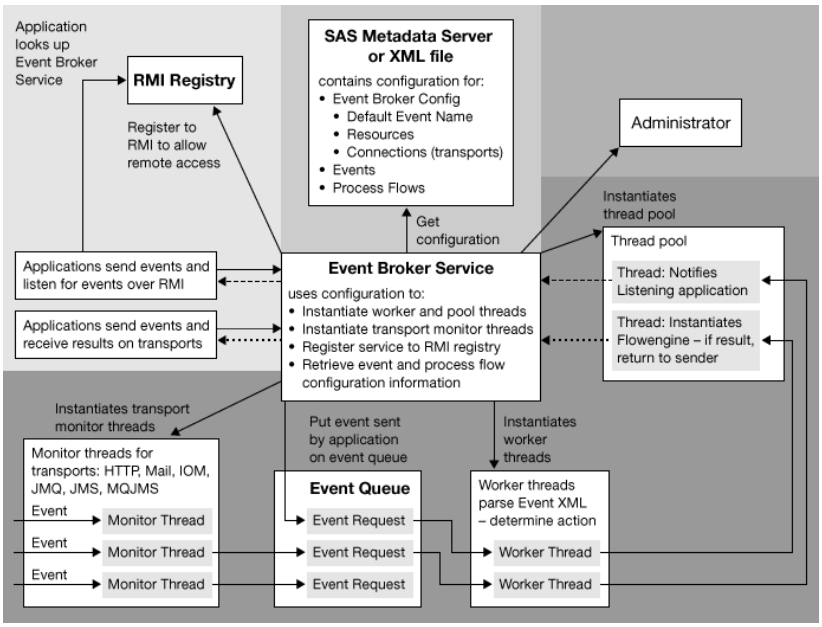
- A structured event is specified as well-formed XML and conforms to the event message specification. (For details about the event message specification, see the SAS Foundation Services class documentation for the Event Broker Service at **`http://support.sas.com/rnd/javadoc/93`**.) It contains information such as the name of the event, the associated properties, and the message body.
- An unstructured event must also be specified as well-formed XML. However, it does not conform to the event message specification. For unstructured events, the entire event is parsed as the message body.

  Note: The Event Broker Service handles unstructured events only if default event handlers have been configured.

For details about the event message specification, see the class documentation for **`com.sas.services.events.broker`** in the SAS Foundation Services class documentation at **`http://support.sas.com/rnd/javadoc/93`**.

The following figure shows the components of the Event Broker Service:

Figure 5.1 *Components of the Event Broker Service*



The Event Broker Service works as follows:

1. **Listens for incoming events via transports or applications.** The Event Broker Service can monitor for and receive events via the following transports:

   RMI
   : If the RMI transport is enabled, the Event Broker Service registers itself to one or more RMI registries.

     To enable the event broker service to be accessed via RMI, you must enable remote access in the service configuration. Enabling remote access registers the service to the RMI Registry. Remote access to the Event Broker Service enables the following:

     - Java clients that are **sending** can use the appropriate RMI registry to locate the Event Broker Service in order to send events.
     - Java clients that are **listening** can use the appropriate RMI registry to locate the Event Broker Service and register to listen for particular events.

   HTTP
   : listens for events that are sent from HTTP clients. Clients can also be SOAP-enabled.

   JMS
   : listens for events that are sent from any JMS-compliant messaging client. This transport uses administered objects to isolate client applications from the proprietary aspects of a provider. When you configure this transport, you specify whether the administered objects are on the local file system or an LDAP directory server. The transport then uses JNDI to look up the administered objects on the local files system or LDAP directory server.

   MQJMS
   : listens for events that are sent from WebsphereMQ (formerly MQSeries) messaging clients.

JMQ
listens for events that are sent from SunONEMQ (formerly iPlanet Message Queue) messaging clients.

Mail
listens for events that are sent to IMAP or POP3 mail servers.

IOM
listens for events that are sent from SAS servers.

2. **Determines the event name to use for event configuration information**. The Event Broker Service parses the event XML to determine the event name (or names if a naming hierarchy is used) to use for event configuration information. If an unstructured event is received, the Event Broker Service uses the service configuration information to map the unstructured event to a default event name.

3. **Forwards the event to the appropriate event handling agents, based on the configured event type.** The Event Broker Service uses configuration information defined for the event name or default event names to determine appropriate actions to take for the event.

   For a broadcast event type, the Event Broker Service notifies all handling agents (process flows and listening applications) of the event as follows:

   - If an application is a registered listener for an event, the Event Broker Service notifies the listening application of the event.
   - If the event configuration contains process flows, the Event Broker Service instantiates a flow engine for each configured process flow in order to process the event message.

   For a request/response event type, the Event Broker Service notifies only one handling agent (listening application or process flow) as follows:

   - If an application is a registered listener for an event, the registered listener has precedence over a process flow. Only one process flow can be defined for request/response types. Therefore, the Event Broker Service forwards the event to the listening application.
   - If the event configuration contains a process flow and there is no application that is a registered listener, the Event Broker Service instantiates a flow engine to process the event message.

4. **Sends a response based on the event response type**. The Event Broker Service uses the event configuration to determine whether to send a response to an event.

   - If the event sender does not require a reply, the event request should specify a response type of **none** or **ack** (acknowledge). To configure an event for no response or acknowledge, you specify **broadcast** as the event type. For acknowledge response types, the Event Broker Service sends an acknowledge receipt to the event sender.
   - If the event sender requires a reply, the event request should specify a response type of **result**. To configure an event for a response, you specify **request/response** as the event type. For request/response types, the Event Broker Service sends a response to the event sender.

   Note: Unstructured event requests are automatically assigned a response type of **none**. Therefore, for event definitions that are used to handle unstructured event requests, you must configure the response type as **broadcast**.

   Note: An event is completely qualified by its name and type. Therefore, the Event Broker Service views events as separate events if they are sent or configured with

the same name, but different event types. For example, if you send an event named **`AlertHigh`** with a response type of **none** to an event broker that contains an event definition named **`AlertHigh`** that is configured as a **request/response** type of event, then an error is returned.

Applications can send and receive events using either of the following:

- transport monitors
- RMI

Understanding Events and Process Flows

Overview of Events

The Event Broker Service configuration enables you to configure one or more events. When an event is received, the Event Broker Service maps the event name to a configured event name. If an unstructured event is received, the Event Broker Service maps the unstructured event to a configured default event name.

An Event configuration consists of the following information:

name
: the name of the event in the incoming XML request maps to the configured event name. You can also name events so that they are part of a naming hierarchy. Events in a naming hierarchy are separated by a period. For example: **`Animals`**, **`Animals.Dogs`**, **`Animals.Dogs.Retriever`**. Naming hierarchies are handled differently based on the event type:

  - If a broadcast event for **`Animals.Dogs.Lab`** is received, the event is delivered to all handling agents (process flow or application) that are registered for **`Animals.Dogs.Lab`**, **`Animals.Dogs`**, and **`Animals`**.
  - If a request/response event is received, it is delivered to a single handling agent. If the incoming request contains an event name that does not exactly match an event name in the Event Broker Service configuration, the naming hierarchy is searched for the best possible event name match that is also configured as a request/response event type.

type
: events can be one of the following types:

  - **Broadcast**, where a notification is sent to all handling agents, and either no response or an acknowledge receipt, is sent to the originating client.
  - **Request/Response**, where notification is sent to one handling agent and a response is sent to the originating client.

    Configure the event type as follows:

    - If the incoming XML request specifies a response type of **none** or **ack** (acknowledge), the event sender does not require a reply. To configure an event for no response or acknowledge, you specify **Broadcast** as the event type. For unstructured events, specify **Broadcast** as the event type.
    - If the incoming XML request specifies a response type of **result**, the event sender requires a reply. To configure an event for a response, specify **Request/Response** as the event type.

Note: If the event configuration does not match the incoming event request response type, then an error is returned (**`Event not configured`**).

- **Security:** you can specify different security attributes for each event:
  - To authenticate and authorize the sender's credential, select the **Check sender's authorization**. If you select the **Check sender's authorization** property, then the event's process flows will not run unless the sender's credentials are successfully authenticated by the SAS Metadata Server's authentication provider and then authorized by the SAS Metadata Server's authorization facility as having the **Execute** permission for the event.

    Note: The sender's event request must contain the sender's user ID and password, and can also contain the authentication domain. You can configure a default authentication domain in the configuration for the User Service (see "Additional Security Configuration" on page 37). If you configure a default authentication domain in the User Service, then the sender is not required to specify the authentication domain in the event request.
  - To run event process flows under a particular identity, you must configure the events to run under one of the following:
    - the sender's identity
    - the broker's identity

      Note: You can configure event process flows to run under the broker's identity only if the Event Broker Service is deployed using a SAS Metadata Server (instead of an XML file) as the metadata source.
    - an identity that you supply in the configuration

    You can also specify that the event run with no security.

The following table summarizes information about the incoming event request/response type and configured event type.

Table 5.1 *Event Types*

Event Request/ Response Type	Configured Event Response Type	Event Notifications	Event Response
none	Broadcast	Notification sent to all process flows configured for the event and all listening applications registered for the event.	No response sent.
ack	Broadcast	Notification sent to all process flows configured for the event and all listening applications registered for the event.	Acknowledge receipt sent to the event sender.

Event Request/ Response Type	Configured Event Response Type	Event Notifications	Event Response
result	Request/Response	Notification sent to only one handling agent (listening application or process flow). If there is a listening application, it takes precedence over the process flow.	Response sent to the event sender.

Additional Security Configuration

To set up security for sender's credentials or event process flows, you must do the following:

- Use the User Manager plug-in to SAS Management Console to define user or group identities in the SAS Metadata Repository.
- Create, configure, and deploy the User Service (of the SAS Foundation Services). You must configure and deploy the User Service as part of the Event Broker Service's service deployment. The User Service must be available to the Event Broker Service at run time.

  To authenticate users, the User Service requires an appropriate login module configuration file. In addition, other Java 2 policy and JAAS policy files might be required. For example, to run an event's process flows under a particular security context, you must set up subject-based security with the JAAS policy configuration file in order to restrict access to the appropriate resources. For details about required User Service configuration, see the SAS Foundation Services class documentation at `http://support.sas.com/rnd/javadoc/93`.

  For details about additional User Service configuration in the Foundation Services Manager, see "Modifying the Session and User Service Configurations" on page 49 .

In addition, to set up authorization for sender credentials, you must grant the sender the Execute permission for the event. To grant the Execute permission, perform the following steps:

1. Use the Authorization Manager plug-in to SAS Management Console to define the Execute permission.
2. From the Foundation Services Manager, open the event properties.
3. On the event's **Authorization** tab, click **Add** to add the appropriate user or group for the sender.
4. Also on the event's **Authorization** tab, select the sender's user or group identity and grant the Execute permission.

After you define an event, you can define your process flows.

Overview of Process Flows

Process flows are used to process event messages. Process flows contain process nodes, which contain logic to process messages, and message nodes, which encapsulate the inputs and outputs for the process nodes.

For broadcast events, you can configure one or more process flows for an event. For request/response events, you can configure only one process flow for an event.

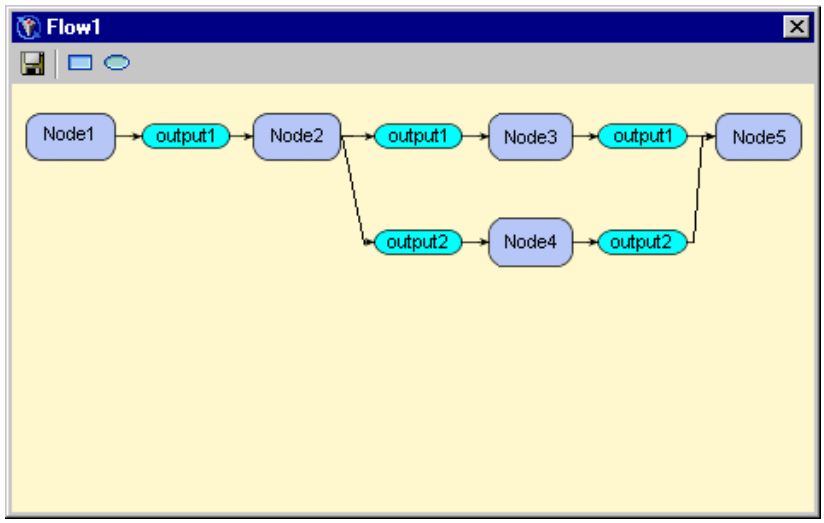
You can configure a process flow by using the Process Flow Editor to define a Process Flow Diagram (PFD). A process flow configuration consists of the following elements:

- **Name and description:** the process name and an optional description.
- **Process nodes:** a process node is a Java class that can have one or more inputs and outputs. You diagram these inputs and outputs as message nodes. When an event is received and a process flow needs to be instantiated for the event, a run-time flow engine is instantiated. The run-time flow engine calls a process node by instantiating the Java class associated with that node. Currently, all process nodes are executed synchronously. A process node configuration consists of the following:
  - **A name and description:** the process node name and an optional description.
  - **A class:** the Java class is used to instantiate the process node. You can then generate the skeleton for the class, define your logic for the class, and compile the class.
  - **Attributes for the class:** attributes are name/value pairs for the class.

  Note: If a process node has no predecessors, then it is the starting node for the process flow. Each process flow can have only one starting node.
- **Message nodes:** a message node encapsulates the outputs and inputs to process nodes in a process flow. A message node configuration consists of the following:
  - **Name and description:** the message node name and an optional description.
  - **Details:** details specify whether a message is required from the previous process node in order to make a process node eligible for firing.

The following display shows an example of a portion of a process flow diagram:

Display 5.1 *Process Flow Diagram*



Modifying an Event Broker Service Configuration

After you create an Event Broker Service in your service deployment, you can modify its service configuration.

To modify the Event Broker Service configuration, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, locate the Event Broker Service that you want to modify.
3. Right-click the service that you want to modify, and select **Properties** from the pop-up menu. The object's properties are displayed.
4. Select the **Service Configuration** tab and click **Configuration**. The EventBroker Service Configuration window appears.
5. On the **Defaults** tab, enter the **Default event name** to use for unstructured events.
6. On the **Resources** tab, enter the event management and thread pool information for the service.
7. On the **Connections** tab, specify an **Administrator Port** and click **Insert** to create a new transport, or select a transport and click **Edit** to edit a transport's properties. For details about creating and editing transports, see the Foundation Services Manager help. When you are finished editing a transport, click **OK**.
8. To enable a service for remote access, perform the following steps:
    a. On the **Connections** tab, select the RMI_Transport, and click **Edit** to edit the RMI transport's properties.
    b. On the **General** tab, to configure a new default event name for RMI transports that overrides the default Event Broker Service event name, enter a **Default event name**.
    c. On the **RMI Details** tab, select the **Enable remote clients to access service capabilities** check box and click **Service Names** to define a new named service. When you are finished creating named services and editing the transport, click **OK**.
9. When you are finished creating or editing a transport, click **OK** to save the Event Broker Service configuration to the metadata repository.

After you edit the Event Broker Service configuration, you can select the Event Broker Service in the navigation tree and create event definitions for the Event Broker Service. The event definitions that you create can then be used to hold process flow definitions that you create.

Creating Events and Process Flows

Create a New Event

To create a new event, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.

2. In the navigation tree, locate the Event Broker Service for which you want to define a new event.
3. Right-click on the Event Broker Service, and select **New Event** from the pop-up menu. The New Event wizard appears.
4. Enter a name and an optional description. Click **Next**.
5. Select the type of event that you want to create. Click **Next**.
6. Select the type of security to use when running the event. Click **Finish** to save the event definition.

After you define an event, you can select the event definition in the navigation tree and create process definitions for the event.

Create a New Process and Process Flow

To create a new process and process flow, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, locate the event for which you want to define a new process flow.
3. Right-click on the event and select **New Process** from the pop-up menu. The New Process wizard appears.
4. Enter a name and an optional description. Click **Next**.
5. Select whether the execution for the process is single-threaded or multi-threaded. Click **Finish** to save the process definition.
6. In the navigation tree, right-click the process flow that you just defined and select **Process Editor** from the pop-up menu. The Process Editor appears.
7. From the toolbar, drag the Process Node () into the drawing area of the Process Editor. The New Process Node wizard appears.
8. Enter a name and an optional description. Click **Next**.
9. Enter the class to instantiate for this process node, and then click **Generate skeleton** to generate skeleton code for the class. Click **Compile** to compile the class. Specify the output directory for the code. Click **Next**.
10. (Optional) Add name/value pairs for the process node. When you have finished entering your name/value pairs, click **Finish**. The Process Node definition is saved. You can now create a message node for outputs and inputs.
11. In Process Flow Editor toolbar, drag the message node () into the drawing area. The Process Message wizard appears.
12. Enter a name and an optional description. Click **Next**.
13. Select the **Usage** drop-down list and choose whether the input is required or optional for downstream process nodes. You can also specify the format for the node. Click **Finish** to define the message node.
14. To create a connection between the process node and the message node, position your cursor on the process or message node so that a pencil icon appears. Drag the cursor to the node to which you are making the connection.
15. Create other process nodes, message nodes, and connections as required for the process flow.

16. Click  to save the process flow.

Modifying the Information Service Configuration

Overview of the Information Service Configuration

The Information Service does the following:

- provides a mechanism to perform a federated search of any repositories that a user has a connection to. The term federated means connected and treated as one. The classes in the Information Service package enable the creation of a single filter, which can search disparate repositories (for example, SAS Metadata Repositories and LDAP repositories).
- allows repository-specific searches to be performed, so that efficient searching can be achieved.
- provides a convenient method for fetching an item from a repository by using a URL.
- can be used in conjunction with the User Services and the Authentication Service to authenticate users, create User Contexts, locate servers that the user has access to, and create repository definitions to use in making server connections.

For more information about the Information Service, see **`com.sas.services.information`** in the SAS Foundation Services class documentation at **`http://support.sas.com/rnd/javadoc/93`**.

The Information Service configuration consists of the following items:

protocols
: map the repository protocol to a Java class that implements the **`com.sas.services.information.RepositoryInterface`** interface. When connecting to a repository, the protocol class definition is used to create the new repository object.

repositories
: are persistent storage mechanisms for metadata and content. The repository definitions specify how to connect to the repository and how to allow client software to connect to a repository by name. You must create a repository definition for each repository that your application accesses. (You must also define a repository when using the getPathUrl method of the MetadataInterface.)

repository groups
: identify a set of repositories that can be searched together.

smart objects
: act as wrappers for metadata entries in order to hide the details of repository-specific metadata types. A smart object definition consists of the following:

  - the protocol of the repository that contains the metadata
  - the interface for the smart object
  - the repository-specific type of metadata
  - the action to take in order to implement the object
  - the filter class to use in order to search for this type of object

You can use smart objects to specify implementations (smart object action definition) for one or more repositories. You must specify an implementation (smart object action definition) for at least one repository type. In the smart object action definition, you can also specify a filter to use for implementing different smart objects for the same repository type.

factories
act as wrappers for metadata entries in order to hide the details of repository-specific metadata types. However, with factories, you cannot specify an interface or filter to use when creating the object. In addition, within each factory, you can specify implementations (factory object action definitions) for only one type of repository. A factory definition consists of the following:

- the protocol of the repository that contains the metadata
- the repository-specific type of metadata
- the action to take in order to implement the factory

You must use smart object definitions if you want to specify the following:

- an interface for the object
- a filter to use when implementing the object
- multiple repositories for the actions of an object

Configure the Information Service

To configure the Information Service, perform the following steps:

1. Open SAS Management Console and connect to a metadata repository.
2. In the navigation tree, locate and select the Information Service that you want to modify.
3. Right-click the Information Service and select **Properties**. The Information Service properties window appears.
4. On the **Service Configuration** tab, click **Configuration**. The Information Service Configuration window appears.
5. On the **Protocols** tab, click **New** to add a protocol, or select a protocol and click **Edit** to edit a protocol. Enter the following information:

   Protocol
   specifies the protocol for the information service.

   Class
   lists the fully qualified Java class for the selected protocol. When requesting a connection to a new repository, this class is used in the *connect* method.
6. (Optional) Define a server on the **Servers** tab. Click **New** to add a server, or select a server and click **Edit** to edit a server. Enter the appropriate information for accessing the server. Click **OK**.
7. (Optional) Define repositories on the **Repositories** tab. Click **New**to add a repository, or select a repository and click **Edit** to edit a repository. Enter the appropriate information for accessing the repository, and then click **Finish**.
8. (Optional) Define repository groups on the **Groups**tab. Click **New** to add a repository group, or select a repository group and click **Edit** to edit a repository group. Enter the following information:

Name
specifies the repository group name.

Member Repositories
specifies the repositories that are members of the repository group. Select a repository from the **Available Repositories** panel and click the arrow button to move it to the **Member Repositories** panel.

9. (Optional) Define smart objects on the **Smart Objects** tab. Click **New** to add a smart object or select a smart object and click **Edit** to edit a smart object. Enter the following information:

Name
specifies the smart object type name. This string should exactly match the string that is returned from the smart object implementation's getType() method.

Interface Class
specifies the fully qualified Java interface that objects of this type will implement.

Filter Class
specifies the fully qualified Java class to use to most effectively search for objects of this type. This class probably contains specific extensions to the `com.sas.services.information.Filter` class to make searches more efficient.

Creation Actions
defines how and when objects of this type are created. An action definition contains a protocol, a repository-specific type, a fully qualified Java class for the implementation to instantiate when that type is encountered, and an optional filter to run against an object which it must match for the action to be taken. Click **Add** to define a new action, or click **Edit** to change an existing action and enter the following information:

Protocol
specifies the repository protocol that this action applies to. Select **omi** for SAS Open Metadata Interface, **ldap** for LDAP directory server, or **dav** for WebDAV server.

Type
specifies the repository-specific type to look for when creating this type of object.

Class
specifies the fully qualified Java class to create when encountering this type in the repository.

Filter
specifies an optional filter, which an object must validate against before this action is taken. The format of the filter is

[**association*/]*@attribute*='*value*'

association	specifies the name of an association from the specified repository type; the objects in the association will be tested against the attribute portion of the filter.
attribute	specifies an attribute to test for validation. The attribute can be an attribute on the objects in the association or, if no association is specified, an attribute can be an attribute on the object itself.

value specifies the attribute value to test the object against to be sure it is the correct type.

10. (Optional) Define factory definitions on the **Factories** tab. Select the **Protocol** for the factory, and then click **New** to add a factory, or select a factory and click **Edit** to edit a factory. Enter the following information:

    Type
    specifies the factory types that are associated with the Information Service and the selected protocol. Specify **Class** (a class to generate the smart object), **Constructor** (a constructor for a Java class that implements the smart object), or **Service** (a Foundation Service).

    Action
    specifies the action that is associated with the selected factory. The **Action** table lists the type, class, method, and filter for each action. Click **Add** to define a new action, or click **Edit** to change an existing action and enter the following information:

    Class
    specifies the fully qualified Java class to instantiate for the action.

    Filter
    specifies the fully qualified Java class to use to search for objects of this type. The class probably contains extensions to the `com.sas.services.information.Filter` class to make searches more efficient.

11. (Optional) Define consumers on the **Consumers** tab. For each consumer, specify a name for the consumer, and then select which resources in the service configuration are consumed.

12. Click **OK** to save the Information Service configuration to the metadata repository.

Modifying the Logging Service Configuration

Overview of the Logging Service

The Logging Service enables applications to do the following:

- send run-time messages to one or more output destinations, including consoles, files, and socket connections.
- configure and control the format of information that is sent to a particular destination. Configuration can be performed through static configuration files or by invoking run-time methods that control logging output.
- perform remote logging, which involves sending log messages that are generated in one Java virtual machine (JVM) to another JVM.
- perform logging either by user session or by JVM.

For more information about the Logging Service, see `com.sas.services.logging` in the SAS Foundation Services class documentation at `http://support.sas.com/rnd/javadoc/93`.

When a service deployment is defined and deployed, a base logging configuration is used to determine the appropriate output destinations. However, you can use SAS

Management Console Foundation Services Manager to modify the Logging Service configuration and configure additional logging contexts and output destinations. The Logging Service configuration consists of the following items:

contexts
: specify the name and outputs for a specific logging context. In your application, you code the Logging Service to send information to a specific logging context. (The RootLoggingContext is used for any logging context that is not configured.) When naming the logging context, you can specify the logging context name as part of a naming hierarchy. In a naming hierarchy, the logging context names are separated by a period (for example, **`com.sas.services.event`**). If a call to a logging context named **`com.sas.services.event`** is made and there is no logging context for **`com.sas.services.event`**, then the Logging Service looks for a logging context of **`com.sas.services`**. If there is no logging context for **`com.sas.services`**, then **`com.sas`** is used. When you define a logging context, you associate outputs with the logging context in order to specify where to send logging messages for that particular logging context.

  Note: To associate outputs with a logging context, you must first create the output definition.

outputs
: specify an output destination for the logging messages. The Logging Service can send the log messages to a file, console, or socket.

renderers
: specify a custom renderer that is used to format logging messages for a specific class. The custom renderer enables you to format logging output when you specify an object as the message parameter in a logging call.

Configure the Logging Service

To configure the Logging Service, perform the following steps:

1. In the SAS Management Console navigation tree, locate and select the Logging Service that you want to modify. Right-click on the Logging Service and select **Properties**. The Logging Service Properties window appears.
2. Select the **Service Configuration** tab and click **Configuration**. The Logging Service Configuration window appears.
3. On the **Outputs** tab, click **New** to add an output, or select an output and click **Edit** to edit an output. Enter the following information:

   ID
   : specifies the name or identifier for the output.

   Layout Pattern
   : specifies how to format the log message. For details about specifying layout patterns, see "Layout Formats" on page 48 .

   Async
   : specifies whether asynchronous logging is activated.

   Type
   : specifies the output type. Select one of the following:

     Console
     : sends the logging output to the system console.

File
writes the logging output to a file.

Socket
sends the logging output to a network socket.

ARM
sends the logging output to an ARM appender.

- If **Console** is selected, then the window contains these items:

  Target
  specifies the output destination. Valid values are *System.out* or *System.err*.

  ImmediateFlush
  specifies whether the contents of the log are sent out after each logging statement.

- If **File** is selected, then the window contains these items:

  File
  specifies the file to use for output.

  ImmediateFlush
  specifies whether the contents of the log are sent out after each logging statement.

  Append
  specifies whether to append the logging output to the existing output in the file. If **Append** is not selected, then any existing data in the file will be overwritten.

  File Rollover
  specifies the type of rollover that is used for the log. Select one of the following:

  None
  specifies that file rollover is not used.

  Size
  specifies that the log file rolls over (starts a new file) when the file reaches a specific size. The **Maximum Backup Index** field specifies the number of backup files that are maintained. The **Maximum File Size** field specifies the maximum file size for the log file before rollover occurs.

  Frequency
  specifies that the log file rolls over (starts a new file) at a specific time interval. The **Frequency Pattern** field specifies the time pattern. For more information about frequency patterns, see the Foundation Services Manager Help.

- If **Socket** is selected, the window contains these items:

  Host
  specifies the socket host (machine name).

  Port
  specifies the socket port number.

- If **ARM** is selected, then the window does not contain any additional items.

Click **OK** to return to the Logging Service Configuration window.

4. On the **Context** tab, click **New** to add a context, or select a context and click **Edit** to edit a context. Enter the following information:

Name
specifies the name of the context.

Priority
specifies the priority level of the logging context. The priority levels are

DEBUG
displays the informational events that are most useful for debugging an application.

INFO
displays informational messages that highlight the progress of the application.

WARN
displays potentially harmful situations.

ERROR
displays error events that might allow the application to continue to run.

FATAL
displays very severe error events that will probably cause the application to terminate.

Chained
specifies whether the context is chained. Chaining designates that the log message is processed by both the current context and also by logging contexts higher in the logging context hierarchy.

Outputs
specifies the output destinations for the context. Move the outputs that you want to associate with this context from the **Available** pane to the **Selected** pane.

Click **OK** to return to the Logging Service Configuration window.

5. (Optional) Specify custom renderers for the Logging service. On the **Renderers** tab, click **New** to add a renderer, or select a renderer and click **Edit** to edit a renderer. Enter the following information:

Rendered Class
specifies the fully qualified name of a Java class that you want to associate with a custom renderer.

Note: Any descendents of the rendered class will also be associated with the custom renderer, following the Java class hierarchy.

Rendering Class
specifies a Java class that implements `com.sas.services.logging.RendererInterface`. The rendering class is used to format the output for any logging calls where an object of the specified **Rendered Class** is passed in as the message parameter.

6. (Optional) Specify consumers for the Logging service. On the **Consumers** tab, specify a **Name** for each consumer and then select which resources in the service configuration are consumed.

7. Click **OK** to save the new Logging Service configuration to the metadata repository.

Layout Formats

The layout specifies how the output is formatted before it is sent to the output device. The layout is specified as a pattern string. The following table shows the characters available for use within layout pattern strings:

The following table shows the special conversion characters available for use within layout pattern strings:

Table 5.2 *Conversion Characters for Layout Patterns*

Conversion Character	Result
c	Used to output the logging context. The logging context conversion specifier can be followed by **precision specifier**, that is a decimal constant in brackets or braces. The precision specifier specifies the number of rightmost components of the logging context name that will be printed. For example, for the logging context name **a.b.c** the pattern **`%c{2}`** outputs **b.c**. If you do not specify a precision specifier, the logging context name is printed in full.
d	Used to output the date of the logging event. The date conversion specifier can be followed by a **date format specifier** enclosed between braces. For example, **`%d{HH:mm:ss,SSS}`** or **`%d{dd MMM yyyy HH:mm:ss,SSS}`**. If no date format specifier is given, then ISO8601 format is assumed.
l	Used to output location information of the caller that generated the logging event. The location information depends on the JVM implementation, but usually consists of the fully qualified name of the calling method followed by the caller's source, the filename, and line number all within parentheses. The location information can be very useful, but its generation can cause performance issues.
m	Used to output the application-supplied message that is associated with the logging event.
n	Used to output the platform-dependent line separator characters. This conversion character offers similar performance to using non-portable line separator strings such as "\n", or "\r\n". Thus, it is the preferred way of specifying a line separator.
p	Used to output the priority of the logging event.
r	Used to output the number of milliseconds elapsed since the start of the application until the creation of the logging event.
s	Used to output the session ID that is associated with this logging event. The output for this conversion character is an empty string if the Logger being used does not have an associated SessionContext.
t	Used to output the name of the thread that generated the logging event.

Conversion Character	Result
u	Used to output the user name that is associated with this logging event. The output for this conversion character is an empty string if the Logger being used does not have an associated SessionContext, or if that SessionContext does not have an associated UserContext.
%	The sequence %% outputs a single percent sign.

For more information about layout formats, see the log4j documentation at **`http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html`** on the Apache Web site.

Modifying the Session and User Service Configurations

Understanding and Editing the User Service

The User Service enables applications to do the following:

- create, locate, maintain, and aggregate information about users of the SAS Foundation Services.
- store and retrieve User Context objects for sharing between applications. The User Context contains the user's active repository connections, identities, and profile.
- manage and access user profiles. A profile is a collection of name/value pairs that specify preferences and configuration or initialization data for a user for a particular application.
- access group profiles. A group profile specifies preferences and configuration or initialization data for a group of users for a particular application.

For more information, see **`com.sas.services.user`** in the SAS Foundation Services class documentation at **`http://support.sas.com/rnd/javadoc/93`**.

The User Service uses a user context to hold the user's information for connections, identities, and profile. The profile then contains application profile data for the user. The User Service configuration consists of the following:

users
: specify the credentials that are associated with this User Service. The user definition consists of the user ID, password, and authentication domain of the user.

profiles
: contain a collection of name/value pairs that specify preferences and initialization data for a user of an application. The profile definition contains the name of the associated application, where the profile is located, the class and type of the profile, and a filter used to locate the profile.

Configure the User Service

To configure the User Service, perform the following steps:

1. In the SAS Management Console navigation tree, locate and select the User Service that you want to modify. Right-click the User Service, and select **Properties** from the pop-up menu. The User Service properties window appears.
2. Select the **Service Configuration** tab. Click **Configuration**. The User Service Configuration window appears.
3. On the **Authentication** tab, specify the default authentication domain.
4. On the **Users** tab, and click **Add** to add a user, or select a user and click **Edit** to edit a user. Click **Select login** to select a metadata login definition, or enter the following information:

   ID
   specifies the user ID of the user.

   Password
   specifies the password needed for the user to log on to the specified authentication domain.

   Confirm Password
   confirms the password that you specified in the **Password** field.

   Domain
   specifies the authentication domain for which the user ID is valid.

   Click **OK** to return to the User Service Configuration window.
5. On the **Profiles** tab, click **Add** to add a profile, or select a profile and click **Edit** to edit a profile. Enter the following information:

   Application
   specifies the application whose profile is specified.

   Domain URL
   specifies the location of the repository where the application profile is stored.

   Class
   specifies the class associated with the profile.

   Type
   specifies the profile type. If you are NOT using a custom profile class, leave this field blank.

   Filter
   specifies information to help locate the correct profile. If you are NOT using a custom profile class, leave this field blank.

   Click **OK** to return to the User Service Configuration window.
6. (Optional) Enter the following information about the **LDAP** tab:

   People
   specifies the distinguished name (DN) for the context in LDAP that contains user metadata.

   Groups
   specifies the DN for the context in LDAP that contains group metadata.

   Credentials
   specifies the location in LDAP that contains credential information.
7. (Optional) Specify consumers for the User service. On the **Consumers** tab, specify a name for each consumer and then select which resources in the service configuration are consumed.

8. When you are finished adding User Service configuration information, click **OK** to save the User Service configuration to a metadata repository.

Understanding and Editing the Session Service

The Session Service enables applications to do the following:

- create a session context. A session context is a control structure that maintains state information within a bound session, facilitating resource management, and context passing.
- bind objects to a session context.
- use the session context as a convenience container for passing multiple contexts.
- use the session context as a convenience container for passing other services, such as User Services and Logging Services.
- notify bound objects when they are removed from the session context or when the session context is destroyed, so that objects can perform any necessary cleanup.

For more information, see **`com.sas.services.session`** in the SAS Foundation Services class documentation at **`http://support.sas.com/rnd/javadoc/93`**.

When the Session Service initializes, it discovers the Logging Service, and obtains a default logging context. The Session Service then uses the Session Service configuration to determine whether to bind to a user context when creating the root session context:

- If the Session Service deployment configuration specifies a user context name, the Session Service discovers the User Service and obtains the default user context. The Session Service then creates a default root session context that is bound to this default user context.
- If the Session Service deployment configuration does not specify a user context name, then the Session Service creates a default root session context that is not bound to any user context.

Applications can then use the root session context to track shared resources that are global to the application and to obtain the initialized logging context and default user context (if one was specified).

Configure the Session Service

To configure the Session Service, perform the following steps:

1. In the SAS Management Console navigation tree, locate and select the Session Service that you want to modify. Right-click the Session Service and select **Properties** from the pop-up menu. The Session Service properties window appears.
2. Select the **Service Configuration** tab and click **Configuration**. The Session Service Configuration window appears.
3. On the **General** tab, specify the default name for the user context. Specify the time-out for the session context.
4. (Optional) Specify consumers for the Session service. On the **Consumers** tab, specify a name for each consumer and then select which resources in the service configuration are consumed.
5. Click **OK** to return to the Session Service Configuration window.
6. Click **OK** to save the Session Service configuration to the metadata repository.

Index

T

U

W

X