Artix™

Session Manager Guide

Version 4.2, March 2007

Making Software Work Together™

Updated: March 21, 2007

Contents

List of Figures

Preface

What is Covered in this Book

This book describes how to use the Artix session manager.

Who Should Read this Book

This book is intended for use by anyone who wants to use the Artix session manager.

How to Use this Book

This book is divided into the following chapters:

- Chapter 1, Introduction, which gives an overview of the Artix session manager.
- Chapter 2, Configuring and Deploying the Session Manager, which describes how to configure and deploy the Artix session manager.
- Chapter 3, Using the Session Manager from an Artix Client, which describes how to write both a C++ client and a Java client of a session managed service. In addition it covers important migration information about moving from Artix 3 to Artix 4.
- Chapter 4, Using the Session Manager from a non-Artix Client, which describes how to write both a .NET client and an Axis client of a session managed service.

The Artix Documentation Library

For information on the organization of the Artix library, the document conventions used, and where to find additional resources, see Using the Artix Library.

CHAPTER 1

Introduction

The Artix session manager enables Web service clients to hold conversations with stateful servers. Client requests are identified as being part of a session and the server can hold state information relating to the client by identifying the requests as part of that client's session. In addition, the session manager controls the number of concurrent clients that can access a Web service and the amount of time allocated to each session.

In this chapter

This chapter includes the following sections:

What is the Session Manager?

Overview

The Artix session manager is implemented as a group of plug-ins that work together to manage the number of concurrent clients allowed to connect to a group of services. An Artix plug-in is a code library that can be loaded into an Artix application at runtime. The session manager plug-ins work together to control how long a client has access to a service before it has to request an extension. In addition, the session manager notifies all registered services of session state changes, including when sessions begin and when they end. This section gives an overview of the session manager's use cases and describes the plug-ins and how they work together in a deployed system.

Use cases

The Artix session manager supports the following use cases:

Limiting the amount of time a client is connected to a service

You can use the Artix session manager to control the amount of time a client has access to a service. This is useful when you do not want clients to have unrestricted access to a service. For example, you might want to limit the amount of time available to complete a request form to five minutes. Clients can request session extensions.

Limiting the number of concurrent client connections to a service

You can specify how many concurrent connections are permitted to a service. For example, if your services are running on old hardware you could ensure higher performance by limiting the number of connections to a small number.

Stateful services

You can write services that store state data across multiple invocations. This is possible because clients of session managed services include identity details with each invocation. Using the session manager's callback mechanism, you can destroy any state information for a client once the client's session expires.

How the session manager works

Using a developer assigned group name, Artix servers register during start-up with the session manager. The session manager maintains a list of servers that register under the same group name. Servers that register under the same group name do not need to offer the same Web service.

Client applications contact the session manager and obtain a session ID for a specific group of servers. Client applications embed the session ID in a context, which must be included with all request to begin, renew, or terminate a session. The session manager sends the clients a collection of endpoint references to all members of the group and the client determines what Web service is represented by each reference and uses the appropriate reference to instantiate a proxy and invoke on the Web service. The client includes the session ID with each invocation.

Session manager plug-ins

The two main session manager plug-ins are:

Session manager service plug-in (`session_manager_service`)	This is the central service plug-in. It accepts and tracks service registration, hands out sessions to clients, accepts or denies session renewal, and notifies session endpoint managers of session state changes, including when sessions begin and when they end.
Session endpoint manager plug-in (`session_endpoint_manager`)	This is the portion of the session manager that resides in a registered service. It registers its location with the service plug-in, and accepts or rejects client requests based on the validity of their session headers.

The session manager also includes a simple policy plug-in:

Session manager simple policy plug-in (`sm_simple_policy`)	This provides control over the allowable duration for a session and the maximum number of concurrent sessions allowed for each group.

The simple policy plug-in is an implementation of the Artix session manager's `SessionManagementPolicyCallback` interface. You can create your own session policies by implementing this interface. For more detail, see "Implementing your own Policy Plug-In" on page 29.

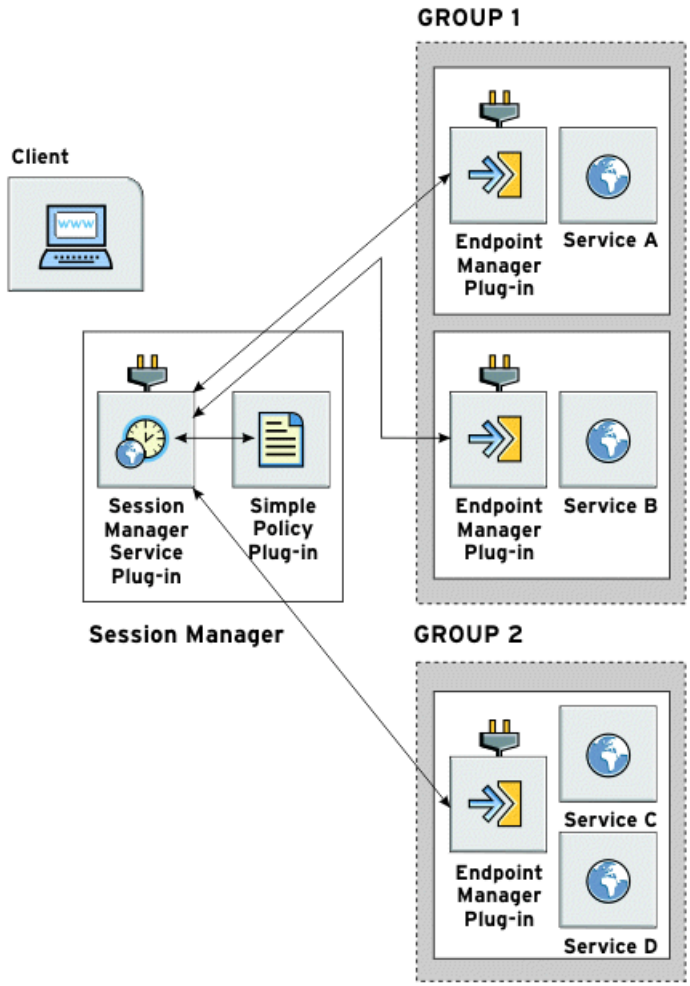
How do the plug-ins interact?

Figure 1 on page 13 shows how the session manager plug-ins are deployed in an Artix system. The session manager service plug-in and the simple policy plug-in are both deployed into the same Artix bus process.

In this example, these plug-ins are deployed in the Artix container. Although they can be deployed in any Artix process, the recommended approach is to use the Artix container. The session manager service plug-in and the simple policy plug-in interact to ensure that the session manager does not hand out sessions that violate the policies established by the simple policy plug-in. The simple policy plug-in makes all the decisions on which sessions are permitted. The session manager service queries this policy on all decisions. Artix provides a default implementation in the simple policy plug-in. You can, however, also write your own policy plug-in.

The endpoint manager plug-ins are deployed into the server processes that contain session managed services. A process can host two services (for example, *Service C* and *Service D* in Figure 1 on page 13), but the process can have only one endpoint manager. The endpoint manager plug-ins are in constant communication with the session manager service plug-in to report on endpoint health. They also receive information on new sessions that have been granted to the managed services, and check on the health of the session manager service.

Figure 1: *Session Manager Plug-ins*



What are sessions?

The session manager controls access to services by handing out *sessions* to clients that request access to the services. A session is a pass that provides access to the services in a specific group for a specific amount of time.

For example, the following process is used when a client application wants to use the services in a group named `sales`:

1. The client application asks the session manager for a session with the `sales` group.
2. The session manager checks and see if the `sales` group has an available session and, if so, it returns a session ID and the list of `sales` service references to the client.
3. The session manager notifies the endpoint managers in the `sales` group that a new session has been issued. It also supplies a new session ID, and a duration for which the session is valid.
4. When the client makes requests on the services in the `sales` group, it must include the session information as part of the request.
5. The endpoint manager for the services checks the session information to ensure it is valid. If it is, the request is accepted. If it is not, the request is rejected.
6. If the client wants to continue using the `sales` services beyond the duration of its session, the client must ask the session manager to renew its session before the session expires.
7. Lastly, when a client's session has expired, it must request a new one.

What are groups?

The Artix session manager does not pass out sessions for each individual service that is registered with it. Instead, services are registered as part of a *group*, and sessions are handed out for the group. A group is a collection of services that are managed as one unit by the session manager. While the session manager does not specify that the services in a group must be related, it is recommended that the endpoints have some relationship.

A service's group affiliation is controlled by the configuration scope in which it is run. To change a service's group, edit the following value in the process configuration scope:

```
plugins:session_endpoint_manager:default_group
```

This specifies the default group name for the services instantiated by the server.

Set up steps

You set up the server side of the session manager using configuration. You do not need to write any dedicated server code. See "Configuring and Deploying the Session Manager" on page 19 for more detail.

Session manager enabling a client requires dedicated coding. See "Using the Session Manager from an Artix Client" on page 35 and "Using the Session Manager from a non-Artix Client" on page 55 for details.

Demonstrations

Artix includes a number of session manager demonstrations, which are located in the following directory of your Artix installation:

```
InstallDir/artix/Version/demos/advanced/session_management
```

For details on how to run the demos, see the `README.txt` file located in this directory.

Session Manager WSDL Contract

Overview

The session manager service is described in the `session-manager.wsdl` contract, which defines the public interface through which the service can be accessed either locally or remotely. A copy of the session manager WSDL contract is stored in the following directory of your Artix installation:

```
InstallDir/artix/Version/wsdl/session-manager.wsdl
```

The session manager WSDL file defines two port types:

- SessionManager port type
- SessionEndpointManager port type

SessionManager port type

The `SessionManager` port type includes operations through which a server process registers and deregisters its endpoint manager and endpoints with the session manager. In addition, it includes operations through which client applications can manage sessions and retrieve a collection of references to all server endpoints registered under a common group name. As an Artix developer you need only understand and use the following operations:

- `beginSession`—a request-response operation used by a client process to initiate a session. If the request to initiate a session is rejected, the session manager returns a `BeginSessionFault`.
- `renewSession`—a request-response operation used by a client process to renew a session. It the request to renew is rejected, the session manager returns a `RenewSessionFault`.
- `endSession`—a oneway operation used by a client process to terminate a session.
- `getAllServiceEndpoints`—a request-response operation used by a client process to obtain the collection of endpoint references belonging to a specific group. If the request is rejected, the session manager returns the `GetAllEndpointsFault`.

SessionEndpointManager port type	The `SessionEndpointManager` port type includes operations through which the session manager communicates session related events to the session endpoint manager associated with a registered service. As an Artix developer, you do not need to use the operations included in this port type.
Binding and protocol	The session manager is accessed through the SOAP binding and over the HTTP protocol.

CHAPTER 2

Configuring and Deploying the Session Manager

This chapter explains how to configure and deploy the session manager.

In this chapter

This chapter discusses the following topics:

Deploying the Session Manager

Overview

The Artix session manager is implemented using Artix plug-ins. This means that any Artix application can host the session manager's core functionality by loading the `session_manager_service` plug-in. However, it is recommended that you deploy the session manager using the Artix container.

This section describes how to configure and deploy the session manager using the Artix container. It also explains how you can deploy the session manager using dynamic port allocation or using a fixed port, and how you can use the container service to shut down a running session manager.

If you are new to Artix configuration and deployment

If you are new to Artix configuration and deployment, you should read the introductory chapters of the Configuring and Deploying Artix Solutions guide.

Artix container

The Artix container is an executable, `it_container`, that provides a basic environment in which to run Web services. Service implementations are loaded into the container as plug-ins.

For more information on the Artix container, see the container chapter in the Configuring and Deploying Artix Solutions guide.

Demo configuration file

The session manager demo includes an example session manager configuration file, called `session_management.cfg`, which is located in the following directory of your Artix installation:

```
InstallDir/artix/Version/demos/advanced/session_management/etc
```

The configuration examples given in this chapter are taken from this file.

Configuring the session manager to run in the container

To configure the session manager service, ensure that the `session_manager_service` plug-in is included in the session manager service configuration scope, for example:

```
session_management {
  ...
   sm_service{
      orb_plugins = ["xmlfile_log_stream", "wsdl_publish",
   "session_manager_service", "sm_simple_policy"];
  ...
    };
```

The `session_manager_service` plug-in implements the session manager service functionality.

In this example the `sm_simple_policy` plug-in is also included in the `orb_plugins` list. If you want to customize settings for this policy, see "Configuring the Simple Policy Plug-in" on page 28.

You can write your own session management policy plug-in and, by adding it to the `orb_plug-ins` list, configure the session manager to use it. For more detail see "Implementing your own Policy Plug-In" on page 29.

If you do not specify a policy plug-in, the `sm_simple_policy` plug-in is loaded automatically by the session manager service.

Configuring a dynamic port

By default, the session manager is configured for deployment on a dynamic port. In the default session manager WSDL contract, the addressing information is as follows:

Example 1: *Session Manager Service on Dynamic Port*

```
<service name="SessionManagerService">
  <port name="SessionManagerPort" binding="sm:SessionManagerBinding">
    <soap:address
   location="http://localhost:0/services/sessionManagement/
             sessionManagerService"/>
  </port>
</service>
```

The highlighted part shows the address. The `localhost:0` port means that when you activate the session manager service, the operating system assigns a port dynamically on startup.

Because the port is assigned dynamically, you must ensure that your clients obtain a reference to the updated contract when it is assigned a port.

For details of using the Artix locator to do this, see the Artix Locator guide.

Configuring a fixed port

There are two ways of configuring the session manager for deployment on a well-known fixed port. You can either edit the default `session-manager.wsdl` contract, or you can create a new `session-manager.wsdl` contract for your application.

Editing the default session manager contract

To edit the default `session-manager.wsdl` contract, perform the following steps:

1. Open the `session-manager.wsdl` contract in any text editor. It is located in the following directory of your Artix installation:

```
InstallDir/artix/Version/wsdl/session-manager.wsdl
```

2. Edit the `soap:address` attribute at the bottom of the contract to specify the correct address. Example 2 shows a modified session manager service contract entry. The highlighted part has been modified to point to the desired address.

Example 2: *Session Manager Service on Fixed Port*

```
<service name="SessionManagerService">
  <port name="SessionManagerPort" binding="sm:SessionManagerBinding">
    <soap:address
   location="http://localhost:8080/services/sessionManagement/session
   ManagerService"/>
  </port>
</service>
```

Creating a new session manager contract

To create a new `session-manager.wsdl` contract, perform the following steps:

1. Copy the default `session-manager.wsdl` contract to another location, and open it in any text editor.
2. Edit the `soap:address` attribute at the bottom of the contract to specify the correct address. Example 2 shows a modified session manager service contract entry. The highlighted part has been modified to point to the desired address.
3. In your configuration file, in the application's scope, add a new `bus:initial_contract:url:sessionmanager` variable that points to your edited WSDL contract. For example:

```
bus:initial_contract:url:sessionmanager =
    "c:\myapp/wsdl/session-manager.wsdl";
```

The default `bus:initial_contract:url:sessionmanager` variable is in the Artix global scope, which ensures that every application has access to the contract. Specifying a new contract in your application scope overrides the global session manager contract for your application.

Configuring a range of ports

You can also limit the range of ports that the session manager is deployed on by specifying a range of ports for the session managers SOAP or HTTP address. Example 3 shows a modified session manager contract entry. The highlighted part specifies the desired range of ports.

Example 3: *Session Manager Port Range*

```
<service name="SessionManagerService">
  <port name="SessionManagerPort" binding="sm:SessionManagerBinding">
    <soap:address
   location="http://localhost:11000-11100/services/sessionManagement/sessionManagerService"/>
  </port>
</service>
```

When the session manager has been correctly configured, it can be started like any other application. The only difference is that the session manager must be started before any servers that need to register with it.

Deploying the session manager using the container

To deploy the default session manager in the container, perform the following steps:

1. Run the session manager in the Artix container; for example:

```
it_container -ORBname demos.session_management.sm_service
   -ORBdomain_name session_management -ORBconfig_domains_dir
   ../../etc -publish
```

   - `-ORBname` specifies the configuration scope under which the container runs the session manager.
   - `-ORBdomain_name` specifies the name of the configuration file that stores the configuration information.
   - `-ORBconfig_domains_dir` specifies the directory where Artix searches for the configuration file.

2. Ask the container to publish the live version of the session manager WSDL that you use to initialize your clients. For example:

```
it_container_admin -container ../../etc/ContainerService.url
   -publishwsdl -service
   {http://ws.iona.com/sessionmanager}SessionManagerService
   -file ..\..\etc\sessionmanager-activated.wsdl
```

   The above command retrieves the session manager's activated WSDL contract. This is the contract in which `0` ports are dynamically updated with the actual port that the service runs on. In this example, `it_container_admin` writes the contract to the `sessionmanager-activated.wsdl` file in the `etc` subdirectory.

3. Lastly, you must ensure that your clients use the updated WSDL file at runtime.

For more information on the Artix container and its command-line parameters, see the container chapter in the Configuring and Deploying Artix Solutions guide.

Deploying the session manager in the container on a fixed port

Alternatively, you can use the `-port` option to specify that the container runs a service on a fixed port. For example:

```
it_container -port 9000 -ORBname demo.sessionmanager.service
   -ORBdomain_name session_management -ORBconfig_domains_dir
   ../../etc -publish
```

In this example, any services that run in the container, and have default contracts with a port of `0`, will not use port `9000`.

You can manually update the WSDL used by your client to `9000`, or you can publish the WSDL from the container using `it_container_admin` with the `-publishwsdl` option, shown in "Deploying the session manager using the container" on page 24.

Shutting down the session manager

To shut down the session manager, use the Artix container's shutdown option, for example:

```
it_container_admin -shutdown
```

Registering a Server with the Session Manager

Overview

For a server to use the session manager it must register itself with a running session manager. Enabling a server to register itself with the session manager is done through configuration. You do not have to write any special server code. Once registered with a session manager, the services only accept client requests that contain valid session headers. All clients that want to access the services must be written to support session managed services.

Any server hosting services that are to be managed by the session manager must load the `session_endpoint_manager` plug-in. The `session_endpoint_manager` enables the server to register with a running session manager. When a server registers an endpoint with the session manager, the session manager creates an association between the group name under which the server process registered and a reference to the endpoint.

Configuring the server

Add the `session_endpoint_manager` to the plug-ins listed under the `orb_plugins` configuration entry within the configuration scope under which the server process runs. Example 4 shows the configuration scope of a server that hosts services managed by the session manager.

Example 4: *Server Configuration Scope*

```
session_management {
...
  server
  {
    orb_plugins = ["xmlfile_log_stream", "wsdl_publish",
   "session_endpoint_manager"];

    plugins:session_endpoint_manager:default_group="SM_Demo";
  };
...
}
```

In this example, a server loaded into the `server` configuration scope is managed by the session manager at the location specified in your `session-manager.wsdl` contract. Its endpoint manager comes up at the address specified in `session-manager.wsdl`. In this example, by default, all services instantiated by the server belong to the `SM_Demo` session manager group.

Using a copy of session-manager.wsdl

If you are using a copy of the default session manager contract to specify a fixed port, your server configuration must also specify the location of the contract. For example:

```
bus:initial_contract:url:sessionmanager =
   "c:\myapp/wsdl/session-manager.wsdl";
```

This is not necessary if you are using a dynamic port, or have updated the default contract with a fixed port. The Artix global scope `bus:initial_contract:url:sessionmanager` setting is used instead.

Server registration

When a properly configured server starts up, it automatically registers with the session manager specified by the contract pointed to by `bus:initial_contract:url:sessionmanager`.

Configuring the Simple Policy Plug-in

Overview

The Artix session manager provides a simple policy callback plug-in (`sm_simple_policy`). This enables you to control the allowable duration for a session, and the maximum number of concurrent sessions allowed for each group.

Session properties

The simple policy plug-in provides default values for the following session properties:

- Maximum number of concurrent sessions in a given group (default is `1`).
- Minimum allowed timeout for a session (default is `1` seconds).
- Maximum allowed timeout for a session (default is `600` seconds).

You can override these defaults using the following configuration variables:

```
plugins:sm_simple_policy:max_concurrent_sessions
plugins:sm_simple_policy:min_session_timeout
plugins:sm_simple_policy:max_session_timeout
```

All values must be non-negative. You must configure the `max_session_timeout` to be greater than or equal to `min_session_timeout`. A value of `0` means an unlimited timeout.

Implementing your own session management policies

The simple policy callback plug-in is an implementation of the Artix session manager's `SessionManagementPolicyCallback` interface. You can create your own session management policy by implementing this interface. For more detail, see "Implementing your own Policy Plug-In" on page 29.

Implementing your own Policy Plug-In

Overview

You can create your own session management policy plug-in by implementing the `SessionManagementPolicyCallback` interface and packaging it as a plug-in. This section explains how.

Procedure

To create your own session management policy plug-in complete the following steps:

1. Implement the `SessionManagementPolicyCallback` interface, shown in Example 5.

Example 5: *SessionManagementPolicyCallback Interface*

```
class SessionManagementPolicyCallback
{
   public:
      virtual void
      begin_session(
         const IT_Bus::String& group,
         const IT_Bus::String& id,
         const IT_Bus::ULong&  preferred_renew_timeout,
         IT_Bus::ULong&        allocated_renew_timeout
      ) IT_THROW_DECL((SessionCreationException)) = 0;

      virtual void
      renew_session(
         const IT_Bus::String& group,
         const IT_Bus::String& id,
         const IT_Bus::ULong&  preferred_renew_timeout,
         IT_Bus::ULong&        allocated_renew_timeout
      ) IT_THROW_DECL((SessionRenewException)) = 0;

      virtual void
      end_session(
         const IT_Bus::String& group,
         const IT_Bus::String& id
      ) = 0;
};
```

The `SessionManagementPolicyCallback` interface is contained in the `it_bus_services/session_manager_service.h` header file.

2. Write a plug-in. For information on writing a plug-in, see the introductory chapters of the Developing Advanced Artix Plug-ins in C++ guide.
3. Integrate your session manager policy and your plug-in by registering your `SessionManagementPolicyCallback` implementation in your plug-in, as shown in Example 6.

Example 6: *Registering your Session Management Policy*

```
void
MySessionsPolicyBusPlugIn::bus_init(
) IT_THROW_DECL((Exception))
{
    Bus_ptr bus = get_bus();

    m_policy = new MySessionPolicy();

    SessionManagerService::register_policy_callback(bus,
   *m_policy);
}

void
MySessionsPolicyBusPlugIn::bus_shutdown(
) IT_THROW_DECL((Exception))
{
   SessionManagerService::deregister_policy_callback(get_bus());
}
```

The register and deregister policy static methods shown are contained in the `it_bus_services/session_manager_service.h` header file.

4. Deploy your session management policy plug-in with the session manager by listing it in the same `orb_plugins` list as the session manager service, and by providing Artix with the root name of the plug-in library, as shown in Example 7 on page 31.

Example 7: *Deploying your Session Management Policy Plug-in*

```
# Artix domain configuration file
session_management {
  ...
  sm_service{
    orb_plugins = ["xmlfile_log_stream",
   "session_manager_service", "my_policy_plugin_name"];

plugins:my_policy_plugin_name:shlib_name="root_library_name"

};
```

Now when the session manager receives requests for new sessions, your session management policy implementation will be consulted.

Fault Tolerance

Overview

Enterprise deployments demand that applications can cleanly recover from occasional failures. The Artix session manager is designed to recover from the two most common failures:

- Failure of a registered endpoint.
- Failure of the session manager itself.

Endpoint failure

When an endpoint gracefully shuts down, it notifies the session manager that it is no longer available. The session manager removes the endpoint from its list so it can not give a client a reference to a dead endpoint. However, when an endpoint fails unexpectedly, it cannot notify the session manager and the session manager can unknowingly give a client an invalid reference causing the failure to cascade.

To decrease the risk of passing invalid references to clients, the session manager occasionally pings all of its registered endpoint managers to see if they are still running. If an endpoint manager does not respond to a ping, the session manager removes that endpoint manager's references.

You can adjust the interval between session manager pings by setting the `plugins:session_manager:peer_timeout` configuration variable. The default setting is `4` seconds. For more information, see the Artix Configuration Reference.

Service failure

If the session manager fails, all of the references to the registered services are lost and the active services are no longer be registered. After the session manager misses its ping interval, the endpoint managers periodically attempt to reregister with the session manager until they are successful. This ensures that the active services reregister with the session manager when it restarts.

You can adjust the interval between the endpoint manager's pings of the session manager by setting the configuration variable `plugins:session_endpoint_manager:peer_timeout`. The default setting is 4 seconds. For more information, see the Artix Configuration Reference.

Adding SOAP 1.2 Support

Overview

The default `session-manager.wsdl` file shipped with Artix contains a SOAP 1.1 binding and a SOAP 1.1 service. As of release 4.1, Artix supports SOAP 1.2 bindings as well.

If your site requires the use of SOAP 1.2 bindings for communication with the session manager, follow these steps:

1. Make a copy of the default `session-manager.wsdl` file.
2. Edit your copy to include a SOAP 1.2 binding. See the SOAP 1.2 chapter of Writing Artix Contracts for guidelines on adding a SOAP 1.2 binding.
3. Use the `bus:initial_contract:url` configuration variable to point to the location of your edited `session-manager.wsdl` file, or use one of several WSDL publishing methods described in "Accessing WSDL Contracts" in Configuring and Deploying Artix Solutions. For SOAP 1.2 both the session manager and the session endpoint manager need to be updated to a SOAP 1.2 binding; for example:

```
bus:initial_contract:url:sessionmanager =
    "session-manager12.wsdl";
bus:initial_contract:url:sessionendpointmanager =
    "session-manager12.wsdl";
```

SOAP 1.2 considerations

The SOAP 1.2 binding in Artix 4.1 (or higher) supports endpoint references (EPRs) only in the format defined by the WS-Addressing standard, and no longer supports the deprecated proprietary Artix references. Artix's SOAP 1.1 binding supports both EPRs and the Artix references used by Artix 3.0 and earlier.

This means that an Artix 4.1 (or higher) session manager that uses the SOAP 1.2 binding cannot support connections from Artix 4.0 and 3.0 clients, because those versions of Artix do not support SOAP 1.2. Thus, when defining your Artix 4.1 (or higher) session manager, if your site intends to maintain backward compatibility with Artix 4.0 and Artix 3.0

clients, do not also use a SOAP 1.2 binding. The configuration step described in "Artix 4.1 and 4.2 session manager setup for backward compatibility" on page 52 is not compatible with a SOAP 1.2 binding.

CHAPTER 3

Using the Session Manager from an Artix Client

Clients that want to use the Artix session manager must include code dedicated to that task. This chapter outlines how to write an Artix session manager client in Java and in C++. In addition, it describes migration scenarios that deal with how to best migrate Artix 3.x applications to Artix 4.

In this chapter

This chapter discusses the following topics:

Implementing a C++ Client

Overview

Clients that want to make requests on session managed services must be designed explicitly to interact with the Artix session manager and must pass session headers to the session managed services. This section describes how to write a session manager client in C++.

Demonstration code

The code examples in this section are taken from the session manager demo's C++ client code. The C++ client makes a request on a business service that is managed by the Artix session manager. The complete client code can be found in the following directory of your Artix installation:

```
InstallDir/artix/Version/demos/advanced/session_management/
cxx/client
```

Implementing a C++ session client

There are eight steps a client takes when making requests on a session managed service. They are:

1. Instantiate a proxy for the session management service.
2. Start a session for the desired service's group using the session manager proxy.
3. Obtain the list of endpoints available in the group.
4. Create a service proxy from one of the endpoints in the group.
5. Build a session header to pass to the service.
6. Invoke requests on the endpoint using the proxy.
7. Renew the session as needed.
8. End the session using the session manager proxy when finished with the services.

Instantiating a proxy

Before a client can request a session from the session manager, it must create a proxy to forward requests to the running session manager. To do this the client creates an instance of `SessionManagerClient` using the session manager's contract name, `session-manager.wsdl`.

Example 8 shows the C++ code for instantiating a session manager proxy.

Example 8: *Instantiating a Session Manager Proxy—C++*

```
// C++
SessionManagerClient session_mgr;
SessionManagerClient* session_mgr_ptr = &session_mgr;
```

Start a session

After instantiating a session manager proxy, a client can then start a session for the desired service's group using the session manager's `beginSession()` function. The `beginSession()` function has the following signature:

```
// C++
virtual void
beginSession(
    const IT_Bus::String &endpoint_group,
    const IT_Bus::ULong preferred_renew_timeout,
    SessionInfo &session_info
) IT_THROW_DECL((IT_Bus::Exception)) = 0;
```

The `beginSession()` function takes the following input parameters:

- `endpoint_group`—the endpoint group name, which corresponds to the default group name set in the server's configuration scope as described in "Registering a Server with the Session Manager" on page 26.
- `preferred_renew_timeout`—the preferred session duration in seconds. If the specified duration is less than the value specified by the session manager's `min_session_timeout` configuration setting, it will be set to the configured minimum value. If the specified duration is higher than the value specified by the session manager's `max_session_timeout` configuration setting, it will be set to the configured max value.

And the following output parameter:

- `session_info`—a sequence complex type that contains the session id, `session_id`, and the actual assigned session duration, `renew_timeout`.

Example 9 shows the C++ client code to begin a session for the SM_Demo group.

Example 9: *Beginning a Session—C++*

```
// C++
...
IT_Bus_Services::IT_SessionManager::SessionId group_session;

int
main(int argc, char* argv[])
{
    ...
        // Begin a session
        session_mgr.beginSession("SM_Demo", 20, session_info);
        cout << "Begin session invoked" << endl;

        // Retrieve the session ID from the response
        group_session = session_info.getsession_id();
        cout << "Got session!" << endl << endl;
    ...
}
```

Get a list of endpoints in the group

The session manager hands out sessions for a group of services. To get an individual service on which the client can make requests, the client needs to get a list of the services in the group. The session manager proxy's getAllServiceEndpoints() function returns a list of all endpoints registered to the specified group. The getAllServiceEndpoints() function has the following signature:

```
// C++
virtual void
getAllServiceEndpoints(
    const SessionId &session_id,
    ServiceEndpointList &endpoints
) IT_THROW_DECL((IT_Bus::Exception)) = 0;
```

The getAllServiceEndpoints() function takes the following input parameter:

- session_id—the session ID for which you are requesting services (obtained in the previous step).

And the following output parameter:

- `endpoints`—the list of services available. If the group has no services, the list will be empty.

Example 10 shows the C++ code for getting the list of services in a group.

Example 10: *Retrieving the List of Services in a Group—C++*

```
//C++
// Get the endpoints for the session.
IT_Bus_Services::IT_SessionManager::ServiceEndpointList
   endpoint_list;

// Must provide the session ID
// Without a valid session ID, the session manager will refuse
// the request
session_mgr.getAllServiceEndpoints(
    group_session,
    endpoint_list
);
```

Create a proxy for the requested service

The client can use any of the services returned by `getAllServiceEndpoints()` to instantiate a service proxy.

The session manager returns the services in the order the services registered with the session manager. Clients are, therefore, responsible for circulating through the list. Otherwise they will all make requests on only one service in the group. In addition, because the session manager does not force all members of a group to implement the same interface, you might need to have your clients to check each service to see if it implements the correct interface by checking the reference's service name as shown in Example 11 on page 40.

Example 11: *Checking the Service Reference for its Interface—C++*

```
//C++
#include <it_bus/wsaddressing_util.h>

using namespace WS_Addressing;

EndpointReferenceType& endpoint = endpoint_list[0];
QName service_name =
   EndpointReferenceUtil::get_service_qname(endpoint);

if (service_name == "", "SOAPService",
"http://www.iona.com/session_management")

{
  // Instantiate a SOAPService proxy
}
else
{
  // do something else
}
```

Example 12 shows the client code for creating a `GreeterClient` proxy from an endpoint reference.

Example 12: *Instantiate a Proxy Server—C++*

```
// C++
GreeterClient client(endpoint_list[0], bus);
```

Create a session header

Services that are being managed by the session manager will only accept requests that include a valid session header. Example 13 shows how to send the session ID in a header by initializing the `sessionIDContext` header context.

Example 13: *Initialize the sessionIDContext Header Context—C++*

```
// C++
using namespace session_management;
using namespace IT_Bus;
using namespace IT_Bus_Services::IT_SessionManager;
```

Example 13: *Initialize the sessionIDContext Header Context—C++*

```
...
const QName DEMO_SESSION_ID_CONTEXT_NAME(
    "",
    "sessionIDContext",
    "http://ws.iona.com/sessionmanager"
);
...
// The session name and session group must be added to each
// request Without valid entries, the session endpoint manager
// will reject the request
ContextRegistry* registry = bus->get_context_registry();
ContextCurrent& current = registry->get_current();
ContextContainer* request_contexts = current.request_contexts();

AnyType* attr = request_contexts->get_context(
    DEMO_SESSION_ID_CONTEXT_NAME,
    true
);

if (0 == attr)
{
    cerr << endl << "Error : Unable to access Session Context"
         << endl;
    return -1;
}

SessionId* session_attr = dynamic_cast<SessionId*> (attr);

if (0 == session_attr)
{
    cerr << endl << "Error : Unable to cast Session Context"
         << endl;
    return -1;
}
session_attr->setname(group_session.getname());
session_attr->setendpoint_group(
    group_session.getendpoint_group()
);
```

For more details about the context API used in this example, see the *Artix Contexts* chapter of the Developing Artix Applications in C++ guide.

Make requests on service proxy

Once the session information is added to the proxy's port information, the client can invoke operations on the endpoint as it would a non-managed service. If the endpoint rejects the request because the client's session is not valid, an exception is raised.

Renewing a session

If a client is going to use a session for a longer than the duration the session was granted, the client must renew its session or the session will timeout. A session is renewed using the session manager proxy's `renewSession()` function. The `renewSession()` function has the following signature:

```
// C++
virtual void
renewSession(
    const SessionInfo &session_info,
    IT_Bus::ULong &renew_timeout
) IT_THROW_DECL((IT_Bus::Exception)) = 0;
```

The `renewSession()` function takes the following input parameter:

- `session_info`—a sequence complex type that contains the session id, `session_id`, and the preferred session duration, `renew_timeout`.

And the following output parameter

- `renew_timeout`—the actual assigned session duration, in seconds.

If the renewal is unsuccessful, an `IT_Bus_Services::renewSessionFaultException` is raised.

End the session

When a client is finished with a session managed service, it should explicitly end its session. This ensures that the session is freed up immediately. A session is ended using the session manager proxy's `endSession()` function. The `endSession()` function has the following signature:

```
// C++
virtual void
endSession(
    const SessionId &session_id
) IT_THROW_DECL((IT_Bus::Exception)) = 0;
```

Example 14 on page 43 shows how to end a session.

Example 14: *Ending a Session—C++*

```
//C++
cout << "Ending session" << endl;
session_mgr.endSession(group_session);
```

Implementing a Java Client

Overview

Clients that want to make requests from session managed services must be designed explicitly to interact with the Artix session manager and must pass session headers to the session managed services. This section describes how to write a session manager client in Java.

Demonstration code

The code examples in this section are taken from the session manager demo's Java client code. The Java client makes a request on a business service that is managed by the Artix session manager. The complete client code can be found in the following directory of your Artix installation:

```
InstallDir/artix/Version/demos/advanced/session_management/
java/client
```

Implementing a Java session client

There are nine steps a client takes when making requests on a session managed service. They are:

1. Register the type factory for the session manager's context data.
2. Instantiate a proxy for the session management service.
3. Start a session for the desired service's group using the session manager proxy.
4. Obtain the list of endpoints available in the group.
5. Create a service proxy from one of the endpoints in the group.
6. Build a session header containing the session ID to pass to the service.
7. Invoke requests on the endpoint using the proxy.
8. Renew the session as needed.
9. End the session using the session manager proxy when finished with the services.

Each of these steps is covered in detail in the subsections that follow.

Registering the session manager's type factory

Artix uses the context mechanism to pass session information between the session manager, clients, and services. Therefore you must register the session manager's type factory with the bus before making any calls on the session manager or session managed services.

Example 15 shows the Java code for registering the session manager's type factory.

Example 15: *Registering the Session Manager's Type Factory—Java*

```
//Java
// bus obtained earlier
bus.registerTypeFactory(new
        com.iona.ws.sessionmanager.SessionManagerTypeFactory());
```

Instantiating a session manager proxy

Before a client can request a session from the session manager, it must create a proxy to forward requests to the running session manager. To do this the client creates an instance of `SessionManagerClient` using the session manager's contract name, `session-manager.wsdl`.

Example 16 shows the Java code for instantiating a session manager proxy.

Example 16: *Instantiating a Session Manager Proxy—Java*

```
//Java
QName name = new QName("http://ws.iona.com/sessionmanager",
                       "SessionManagerService");
QName portName = new QName("", "SessionManagerPort");

URL wsdlLocation = null;
try
{
  wsdlLocation = new URL(wsdlPath);
}
catch (java.net.MalformedURLException ex)
{
  wsdlLocation = new File(wsdlPath).toURL();
}

ServiceFactory factory = ServiceFactory.newInstance();
Service service = factory.createService(wsdlLocation, name);
SessionManager sessionMgr =
   (SessionManager)service.getPort(portName,
   SessionManager.class);
```

For more information on instantiating Artix proxies, see the *Things to Consider When Developing Artix Applications* chapter, in the Developing Artix Applications in Java guide.

Start a session

After instantiating a session manager proxy, a client can then start a session for the desired service's group using the session manager's `beginSession()` method.

The `beginSession()` method has the following signature:

```
//Java
SessionInfo beginSession(String endpoint_group,
                         BigInteger preferred_renew_timeout);
```

The `beginSession()` function takes the following input parameters:

- `endpoint_group`—the endpoint group name, which corresponds to the default group name set in the server's configuration scope as described in "Registering a Server with the Session Manager" on page 26.
- `preferred_renew_timeout`—the preferred session duration in seconds. If the specified duration is less than the value specified by the session manager's `min_session_timeout` configuration setting, it will be set to the configured minimum value. If the specified duration is higher than the value specified by the session manager's `max_session_timeout` configuration setting, it will be set the configured max value.

And returns the following:

- `SessionInfo`—a sequence complex type that contains the session id, `session_id`, and the actual assigned session duration, `renew_timeout`.

Example 17 shows the Java client code to begin a session for `SM_Demo`.

Example 17: *Beginning a Session—Java*

```
//Java
SessionInfo sessionInfo = null;
String _endpoint_group = "SM_Demo";
BigInteger _preferred_renew_timeout = new
   Java.math.BigInteger("20");
sessionInfo = sessionMgr.beginSession(_endpoint_group,
                                      _preferred_renew_timeout);
```

Get a list of endpoints in the group

The session manager hands out sessions for a group of services. To get an individual service on which the client can make requests, the client needs to get a list of the services in the group. The session manager proxy's `getAllServiceEndpoints()` method returns a list of all endpoints registered to the specified group. The `getAllServiceEndpoints()` method has the following signature:

```
//Java
ServiceEndpointList getAllServiceEndpoints(SessionId
   session_id);
```

The `getAllServiceEndpoints()` function takes the following input parameter:

- `session_id`—the session ID for which you are requesting services (obtained in the previous step).

And returns the following output:

- `endpoints`—the list of services available. If the group has no services, the list will be empty.

Example 18 shows the Java code for getting the list of services in a group.

Example 18: *Retrieving the List of Services in a Group—Java*

```
//Java
ServiceEndpointList endpointList = null;

endpointList =
   sessionMgr.getAllServiceEndpoints(sessionInfo.getSession_id()
   );
```

Create a proxy for the requested service

The client can use any of the services returned by `getAllServiceEndpoints()` to instantiate a service proxy.

Example 19 shows the Java client code for creating a `GreeterClient` proxy from an endpoint reference.

Example 19: *Instantiate a Proxy Server—Java*

```
//Java
EndpointReferenceType[] references =
   endpointList.getEndpointReference();
Greeter greeter = (Greeter)bus.createClient(references[0],
                                            Greeter.class);
```

Create a session header

Services that are being managed by the session manager will only accept requests that include a valid session header. Example 20 shows the Java code for sending the session ID in a header by initializing the `sessionIDContext` header context.

Example 20: *Initialize the sessionIDContext Header Context—Java*

```
//Java
ContextRegistry registry = bus.getContextRegistry();

QName principalCtxName = new QName("", "SessionId");
QName principalCtxType = new
   QName("http://ws.iona.com/sessionmanager", "SessionId");
QName principalMessageName = new
   QName("http://ws.iona.com/sessionmanager", "", "");
String principalPartName = "id";

registry.registerContext(principalCtxName,
                         principalCtxType,
                         principalMessageName,
                         principalPartName);


IonaMessageContext contextImpl =
   (IonaMessageContext)registry.getCurrent();
SessionId sessionId = sessionInfo.getSession_id();
contextImpl.setRequestContext(principalCtxName, sessionId);
```

For more details about the context API used in this example, see the *Using Message Contexts* chapter of the Developing Artix Applications in Java guide.

Make requests on service proxy

Once the session information is added to the proxy's port information, the client can invoke operations on the endpoint as it would a non-managed service. If the endpoint rejects the request because the client's session is not valid, an exception is raised.

Renewing a session

If a client is going to use a session for a longer than the duration the session was granted, the client must renew its session or the session will timeout. A session is renewed using the session manager proxy's `renewSession()` method. The `renewSession()` method has the following signature:

```
//Java
BigInteger renewSession(SessionInfo session_info);
```

The `renewSession()` function takes the following input parameter:

- `session_info`—a sequence complex type that contains the session id, `session_id`, and the preferred session duration, `renew_timeout`.

And the following output parameter:

- `BigInteger`—the actual assigned session duration, in seconds.

If the renewal is unsuccessful, an exception is raised.

End the session

When a client is finished with a session managed service, it should explicitly end its session. This ensures that the session is freed up immediately. A session is ended using the session manager proxy's `endSession()` method. The `endSession()` method has the following signature:

```
//Java
void endSession(SessionId);
```

Example 21 shows the Java code for ending a session.

Example 21: *Ending a Session—Java*

```
//Java
sessionMgr.endSession(sessionId);
```

Migrating from Earlier Versions

Overview

With the release of Artix 4.0 and subsequent releases, the following changes might affect any existing Artix applications:

- Session manager API name changes were made in compliance with the wrapped doc-literal convention.
- Artix switched from using a proprietary reference format to using the standard WS_Addressing endpoint reference format.

If you have existing applications that use the old session manager APIs and the old proprietary reference format, you might want to consider migrating those applications to use the new APIs and WS_Addressing.

For WS_Addressing migration information, see the *Endpoint References* chapter in the Developing Artix Application in C++ guide and/or the *Using Endpoint References* chapter in the Developing Artix Applications in Java guide. This section describes the session manager API migration scenarios.

New session manager API

Artix 4.0 and subsequent releases include a new version of the `session-manager.wsdl` file. The operations contained in this new WSDL file conform with the wrapped doc-literal convention. Specifically:

- The `begin_session()` operation has been replaced with `beginSession()`.
- The `end_session()` operation has been replaced with `endSession()`.
- The `renew_session()` operation has been replaced with `renewSession()`.
- The `get_all_endpoints()` operation has been replaced with `getAllServiceEndpoints()`. The `get_all_endpoint()` operation returns an `EndpointList` of old style `References`. The `getAllServiceEndpoints()` operation returns a `ServiceEndpointList` of WS-Addressing type `EndpointReferenceType`.

The new `session-manager.wsdl` file is located in the following directory of your Artix installation:

```
InstallDir/artix/Version/wsdl
```

In Artix 4.0 and subsequent releases, by default, the session manager resolves its service contract against this `session-manager.wsdl` file and, therefore, supports the new API. The default Artix configuration file, `artix.cfg`, points to the new session manager WSDL file as follows:

```
bus:initial_contract:url:sessionmanager =
   "InstallDir/artix/Version/wsdl/session-manager.wsdl";
```

Migrating to new session manager APIs

If you have an existing application that you want to migrate to Artix 4.0 or higher, you can switch to using the new APIs by changing the following aspects of your application:

- Replace `begin_session()` with `beginSession()`
- Replace `end_session()` with `endSession()`
- Replace `renew_session()` with `renewSession()`
- Replace `get_all_endpoints()` with `getAllServiceEndpoints()`

Using a mixture of old and new session manager APIs

Artix 4.0 and subsequent releases include a second `session-manager.wsdl` file that supports both the old and the new APIs. To use the session manager with Artix 3 clients, you must start the session manager with this `session-manager.wsdl` file. It is located in the following directory of your Artix installation:

```
InstallDir\artix\Version\wsdl\oldversion
```

You can configure the session manager to use this `session-manager.wsdl` file by setting the `bus:initial_contract:url:sessionmanager` configuration variable as follows:

```
bus:initial_contract:url:sessionmanager =
   "InstallDir/artix/Version/wsdl/oldversion/
    session-manager.wsdl";
```

Alternatively, you can set it as a command-line argument when launching a server:

```
-BUSservice_contract
   InstallDir/artix/Version/wsdl/oldversion/session-manager.wsdl
```

Note: The session manager and the endpoints it manages are tightly coupled and, therefore, must be the same version.

Artix 4.1 and 4.2 session manager setup for backward compatibility

The `artix.cfg` file included with Artix 4.1 and 4.2 has a new configuration entry, `bus:non_compliant_epr_format`. The default `artix.cfg` sets this entry by default to `"false"`. This setting allows for greater interoperability between Artix and Web services software from other vendors.

If your site uses a session manager, session manager enabled services, and session manager enabled clients all built with Artix 4.1 or higher, then no further configuration is necessary.

If your site uses a session manager build with Artix 4.1 or higher, with services and clients from Artix 4.0 and 3.0.x, then you must add one configuration entry in your Artix configuration. Add the line to the `session_management.sm_service` scope of the configuration file that controls your instance of the session manager. The line to add is:

```
bus:non_compliant_epr_format = "true";
```

Note: The session manager demos that ship with Artix 4.1 and 4.2 do not have this line added to their `session_management.cfg` files.

For example, the following configuration file extract shows an edited `session_management.cfg` file for the primary session manager demo that allows Artix 3.x and 4.0 clients to connect to and use an Artix 4.1 or higher session manager:

```
demos {
 session_management {

  plugins:xmlfile_log_stream:use_pid = "true";

  client
  {
    orb_plugins = ["xmlfile_log_stream"];
  };

  sm_service
  {
    bus:initial_contract:url:sessionmanager =
   "../../etc/session-manager.wsdl";

    plugins:sm_simple_policy:max_concurrent_sessions = "1";
    plugins:sm_simple_policy:min_session_timeout = "1";
    plugins:sm_simple_policy:max_session_timeout = "600";

    orb_plugins = ["xmlfile_log_stream", "wsdl_publish",
   "session_manager_service", "sm_simple_policy"];
    bus:non_compliant_epr_format = "true";

  };

  server
  {
    orb_plugins = ["xmlfile_log_stream",
   "session_endpoint_manager"];

   bus:initial_contract:url:sessionmanager =
   "../../etc/session-manager.wsdl";

    plugins:session_endpoint_manager:default_group = "SM_Demo";
  };

 };
};
```

Disabling session manager support for Artix 3

When you have all Artix client applications migrated to Artix 4, the backward compatibility feature of the Artix 4 session manager is no longer necessary for your site. However, there is no need to disable the backward compatibility feature, and the Artix 4 session manager performance is not improved by disabling backward compatibility.

If you prefer to disable this feature, you can use a local configuration scope to override the Artix root configuration. In your local scope, set the WSDL path to empty for the Artix 3-compatible version of the session manager, using a line like the following:

```
bus:qname_alias:sessionmanager_oldversion = "";
```

CHAPTER 4

Using the Session Manager from a non-Artix Client

Non-Artix clients can also use the session manager to make requests on managed services. This chapter outlines how to implement a .NET client and an Axis client.

In this chapter

This chapter discusses the following topics:

Implementing a .NET Client

Overview

.NET clients can use the session manager to make requests on managed services, using the `Bus.Services.dll` library. This is because the Artix session manager uses SOAP headers to pass session tokens between clients and services. The session manager also has a number of methods for managing active sessions. The Artix .Net plug-in is Web Services Enhancements 2.0 (WSE 2.0) compliant. Users can enable session by constructing a session filter and appending it to a SOAP output filter using WSE 2.0 APIs. The helper classes included in the `Bus.Services` library simplify working with the session manager by providing native .Net calls to access the session manager. They also handle session renewal and attaching session headers to outgoing requests.

What you need before starting

Before starting to develop a client that uses the Artix session manager you need:

- A means for contacting a deployed Artix session manager. This can be one of the following:
  - An Artix reference
  - An HTTP address
  - A local copy of the session manager WSDL contract
- A locally accessible copy of the WSDL contract that defines the service that you want the client to invoke upon.
- To install WSE 2.0 SP3 before starting an Artix .NET session manager client.

Demonstration code

The code examples in this section are taken from the session manager demo's .NET client code. The .NET client makes a request on a business service that is managed by the Artix session manager. The complete client code can be found in the following directory of your Artix installation:

```
InstallDir\artix\Version\demos\advanced\session_management\
dotnet\client
```

Procedure

To develop a .Net client that uses the Artix session manager do the following:

1. Create a new project in Visual Studio.
2. Right-click the folder for you new project and select **Add Reference** from the pop-up menu.
3. Click **Browse** on **Add Reference** window.
4. In the file selection window browse to your Artix installation and select the `Bus.Services.dll` from the *`InstallDir`*`\artix\`*`Version`*`\utils\.NET` directory.
5. Click **OK** to return to the Visual Studio editing area.
6. Right-click the folder for your new project and select **Add Web Reference** from the pop-up menu.
7. In the **Address:** field of the browser, enter the full pathname of the contract for the service on which you are going to make requests.
8. Click **Add Reference** to return to the Visual Studio editing area.
9. Open the `.cs` file generated for the contract you imported.
10. Locate the class declaration for the service on which you intend to make requests. The class declaration will look similar to that shown in Example 22.

Example 22: *.Net Service Proxy Class Declaration*

```
public class SOAPService :
   System.Web.Services.Protocols.SoapHttpClientProtocol {
```

11. Change the class' base type from `System.Web.Services.Protocols.SoapHttpClientProtocol` to `Microsoft.Web.Services2.WebServicesClientProtocol`. The resulting class declaration will look similar to that shown in Example 23.

Example 23: *.Net Session Managed Proxy Class Declaration*

```
public class SOAPService :
   Microsoft.Web.Services2.WebServicesClientProtocol {
```

Reassigning the service proxy class to the Artix specific base class adds

methods to the proxy that allow it to work with the session manager.

12. Add a new C# class to your project.
13. Add the statement `using Bus.Services;` after the statement `using System;`.
14. Create a service proxy for the Artix session manager by instantiating an instance of the `Bus.Services.SessionManager` class as shown in Example 24.

Example 24: *Instantiating a Session Manager Proxy in .Net*

```
SessionManager sessionManager = new SessionManager
   ("http://localhost:9007/services/sessionManagement/
    sessionManagerService");
```

The constructor's parameter is the HTTP address of a deployed session manager. The `SessionManager` class also has a construct that takes an Artix reference for use with the Artix locator.

15. Create a new Artix session by instantiating an instance of `Bus.Services.Session` as shown in Example 25.

Example 25: *Creating a New Session*

```
Session session = new Session(sessionManager, "SM_Demo", 20);
```

The constructor takes three parameters:

- An instantiated `SessionManager` object.
- A string identifying the group for which the client wants a session; in this example, the group name is `SM_Demo`.
- The default timeout value, in seconds, for the session.

Once the session is created, the session will automatically attempt to renew itself until the session is closed. The client does not need to worry about renewing the session.

16. Get a list of the references for the endpoints that are in the session's

group using the `SessionManager.get_all_endpoints()` function as shown in Example 26.

Example 26: *Getting the Endpoint References*

```
Bus.Services.Types.EndpointReferenceType[] refs =
   sessionManager.getAllServiceEndpoints(sessionId);
```

The `get_all_endpoints()` function takes the session ID of the session and returns an array of Artix references. Each entry in the array contains the endpoint of one member of the group for which the session was requested.

17. Create a .Net proxy for the service on which you are going to make requests as you normally would.
18. Change the value of the proxy's `.Url` member to the SOAP address of one of the Artix references returned from the session manager as shown in Example 27.

Example 27: *Changing the URL of a .Net Service Proxy to Use a Reference*

```
simpleService.Url = refs[0].Address.Value;
```

How you determine which member of the returned array contains the desired endpoint is an implementation detail beyond the scope of this discussion.

19. Instruct the proxy to include the session header in all of its requests by adding a session filter on the proxy output SOAP filters as shown in Example 28.

Example 28: *Setting a Proxy's Session Header*

```
simpleService.Pipeline.OutputFilters.Add(new
   Bus.Services.SessionFilter(session));
```

Once you have made the above call, all requests made by the proxy will contain an Artix session header. The session manager uses the session header to validate the client's requests against the list of valid sessions.

20. Make requests on the service as you would normally.

21. When you are done with the service, end the session by calling `EndSession()` on the `session` object, as shown in Example 29:

Example 29: *Ending a Session*

```
session.EndSession()
```

Note: For a complete list of available classes and methods, see the `docs.xml` file, which is generated during the `Bus.Services` build. It is available in the following directory of your Artix installation:

InstallDir\artix*Version*\utils\.NET

Implementing an Axis Client

Overview

An Axis client can use the session manager to invoke on managed services. The Artix session manager uses SOAP headers to pass session tokens between clients and services. Therefore, when writing an Axis client, you must insert session tokens into SOAP headers programmatically in order to invoke on services managed by session manger.

Demonstration code

The code examples in this section are taken from the session manager demo's Axis client code. The Axis client makes a request on a business service that is managed by the Artix session manager. The complete client code can be found in the following directory of your Artix installation:

```
InstallDir/artix/Version/demos/advanced/session_management/Axis/
   client
```

Axis version

Axis version 1.3 is used in the demo.

Procedure

To develop an Axis client that uses Artix session manager do the following:

1. Generate Axis stub code from the Artix session manager WSDL file as shown in Example 30:

Example 30: *Generating Axis Stub Code for Session Manager*

```
Java  org.apache.axis.wsdl.WSDL2Java ..\etc\session-manager.wsdl
```

The `session-manager.wsdl` file is available in the following directory of your Artix installation:

```
InstallDir/artix/Version/wsdl
```

2. Generate Axis stub code from the WSDL file for the service on which you want your client to invoke, as shown in Example 31:

Example 31: *Generating Axis Stub Code for the Target Web Service*

```
Java  org.apache.axis.wsdl.WSDL2Java
   ..\etc\session_management.wsdl
```

In this example, the `session_management.wsdl` file is part of the session manager demo and describes the business service on which the client ultimately invokes. It is available in the following directory of your Artix installation:

```
InstallDir/artix/Version/demos/advanced/session_management/etc
```

3. Retrieve a session manager service endpoint as shown in Example 32:

Example 32: *Retrieving a Session Manager Service Endpoint*

```
java.lang.String url =
   get_soap_address("../etc/session-manager.wsdl", service,
   port);
java.net.URL endpoint = new java.net.URL(url);
```

4. Instantiate a session manager proxy as shown in Example 33:

Example 33: *Instantiating a Session Manager Proxy*

```
SessionManagerService smsl = new SessionManagerServiceLocator();
SessionManagerBindingStub sm_binding =
   (SessionManagerBindingStub)smsl.getSessionManagerPort
   (endpoint);
```

5. Start a new session as shown in Example 34:

Example 34: *Starting a Session*

```
SessionInfo session_response = null;

session_response = sm_binding.beginSession("SM_Demo", new
   org.apache.axis.types.UnsignedLong(20));
```

6. Retrieve the session ID and all the endpoints as shown in Example 35:

Example 35: *Retrieving a Session ID and the Endpoints*

```
SessionId session_id = session_response.getSession_id();
EndpointReferenceType[] endpoints =
   sm_binding.getAllServiceEndpoints(session_id);
```

7. Retrieve the first endpoint as shown in Example 36:

Example 36: *Retrieving the Business Service Endpoint*

```
EndpointReferenceType epr_ref = endpoints[0];
String url = epr_ref.getAddress().get_value().toString();
java.net.URL simple_endpoint = new java.net.URL(url);
```

8. Insert the session ID into the SOAP header of the Axis client request as shown in Example 37:

Example 37: *Inserting the Session ID into the Axis Client Request SOAP Header*

```
String ns = "http://ws.iona.com/sessionmanager";
header = new org.apache.axis.message.SOAPHeaderElement(ns, "id",
   session_response.getSession_id());
proper_call.addHeader(header);
```

You must insert the session context into the SOAP header programmatically for each invocation. Otherwise, the invocation will fail.

9. Invoke on the endpoint, as shown in Example 38:

Example 38: *Invoking on the Business Service*

```
String _return = (String)proper_call.invoke(new
   java.lang.Object[] {});
```

10. End the session, as shown in Example 39:

Example 39: *Ending the Session*

```
sm_binding.endSession(session_id);
```

Index