Oracle® Fusion Middleware

Developing Applications and Introspection Plug-ins for Oracle Virtual Assembly Builder

12*c* (12.1.2)

E29472-01

June 2013

Oracle Fusion Middleware Developing Applications and Introspection Plug-ins for Oracle Virtual Assembly Builder, 12*c* (12.1.2)

E29472-01

Contents

Preface

This book details development topics about Oracle Virtual Assembly Builder. This Preface includes the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

The intended audience is Oracle Virtual Assembly Builder developers who create applications leveraging the following development capabilities:

- Using the Web Service API to create PaaS-enabled applications.
- Using the Introspector Plug-in SDK to create introspection plug-ins.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the documentation set:

- *Installing Oracle Virtual Assembly Builder*
- *Using Oracle Virtual Assembly Builder*
- *Release Notes for Oracle Virtual Assembly Builder*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview of Oracle Virtual Assembly Builder Deployer

The following sections introduce the Web Service API and its development processes:

- Section 1.1, "Introduction"
- Section 1.2, "Deployment Lifecycle"
- Section 1.3, "Deployer Architecture"
- Section 1.4, "Targets"
- Section 1.5, "Tags"
- Section 1.6, "Metadata-Driven Functionality"

1.1 Introduction

The Oracle Virtual Assembly Builder Deployer is a J2EE application that maintains a repository of *assembly archives*, and manages deployment aspects of the software system contained therein. You create assembly archives in Oracle Virtual Assembly Builder Studio.

The Deployer provides operations for registering the assembly archives to virtualized systems such as Oracle VM and for orchestrating the deployment of the software system defined by the assembly archive.

The assembly archives created by Oracle Virtual Assembly Builder Studio contain information about a software system comprised of multiple, related software stacks which work together to form an application. This system is referred to as an assembly. The assembly archive contains metadata about the assembly and virtual machine templates that are used to instantiate an instance of the assembly in a virtualized environment.

The interface to the Oracle Virtual Assembly Builder Deployer is a Web service which provides operations for uploading assembly archives to its repository, registering the assembly archive with the virtualization system and managing deployment instances for the system defined in the assembly archive.

Oracle Virtual Assembly Builder provides additional interfaces as part of the complete product solution. Specifically, Oracle Virtual Assembly Builder Studio and the `abctl` command-line interface integrate with the Web Services.

As described in Section 1.4, "Targets", Oracle Virtual Assembly Builder Deployer can deploy to targets in Oracle VM. The minor functional differences between these environments are described throughout the documentation.

1.2 Deployment Lifecycle

This section describes the deployment lifecycle including the uploading of assembly archives to the Deployer repository, and the deployment phases of an assembly instance.

1.2.1 Uploading Assembly Archives

Oracle Virtual Assembly Builder Deployer is a Web service which provides operations for uploading assembly archives to its repository.

1.2.1.1 Versioning of Assembly Archives

The Deployer repository allows you to keep multiple versions of an assembly archive in the repository at the same time. When an assembly archive is uploaded, the Deployer will assign a version number to it.

The versioning format increments in whole number units (for example, versions 1, 2, 3, and so on). The latest version is generally considered the default for assembly operations that take a version. If another assembly archive of the same name is uploaded, then it will get the next version.

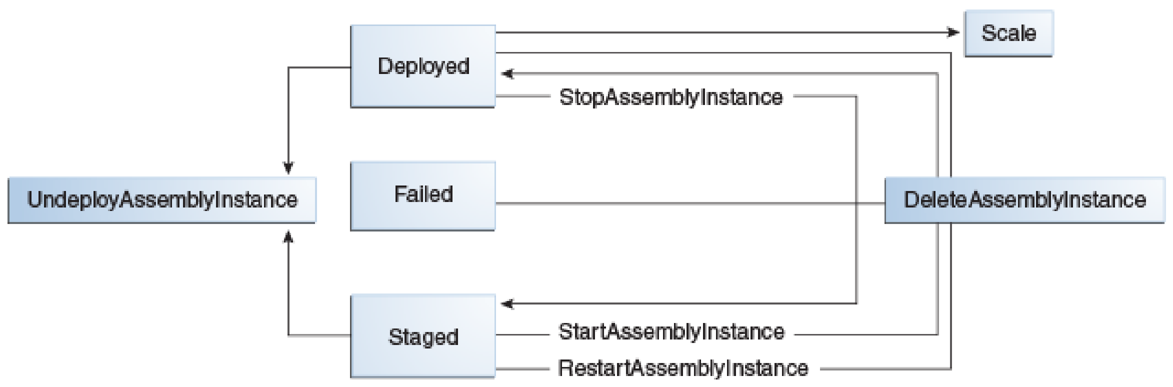
1.2.2 Deployment Phases

An assembly instance is a deployable artifact. You would need to create an assembly instance by selecting an assembly, one of the its deployment plans and the target to which it must be deployed to. `CreateAssemblyInstance` can be used to create the assembly instance.

At deployment time, you choose the assembly instance to be deployed. Deployment of an assembly instance will transition through various phases (Figure 1–1). The phases include: *Staged*, *Deployed*, and *Failed*. Each state allows a subset of operations.

An appliance instance is an instance of an appliance running and/or created in the target virtual environment. When an assembly instance is deployed, you may start and stop the appliance instances, or you may increase or decrease the number of appliance instances associated with that deployed assembly instance. Oracle Virtual Assembly Builder does not monitor the health of the deployed application; it will only inform you of whether or not an assembly is deployed or staged, as well as the success or failure of a deployment-related operation.

Figure 1–1 Deployment Life Cycle



Here is a summary of the assembly instance phases:

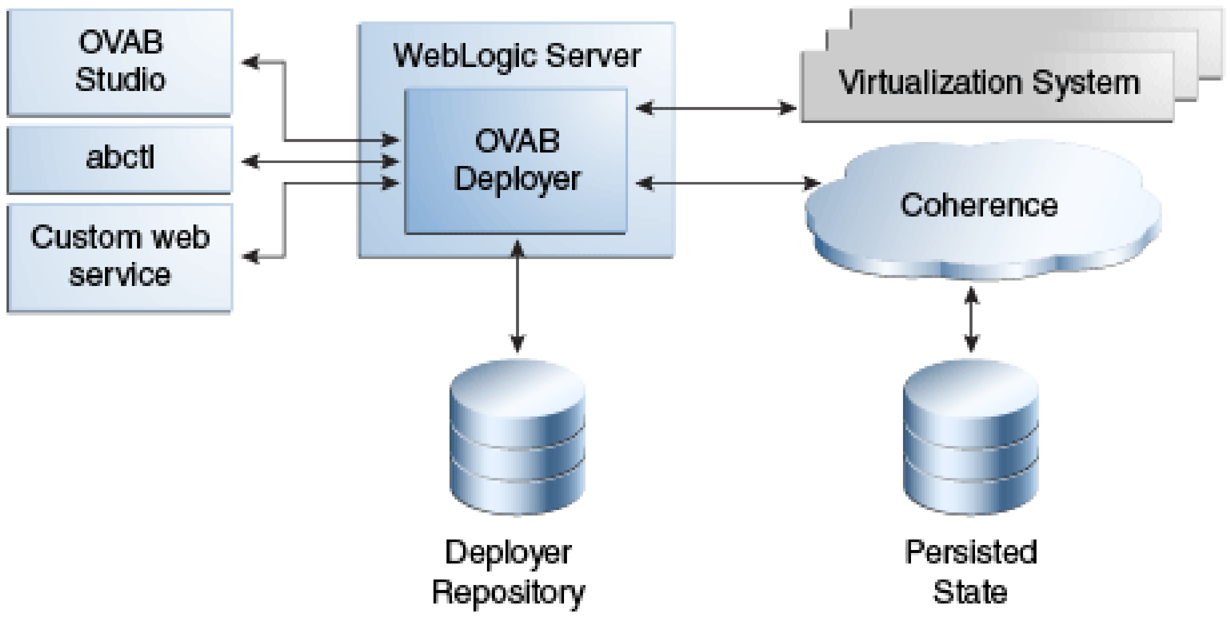
- *Deployed* When the assembly instance is deployed and the operation has successfully completed, it reaches the deployed state. The operations that can be performed on a *Deployed* assembly instance are:
  - *StopAssemblyInstance* This operation will shut down all the running appliance instances for the assembly instance. The assembly instance is transitioned to the *Staged* phase after this operation is completed. It leaves the appliance instances in the virtualized environment so that they can be restarted later.
  - *UndeployAssemblyInstance* This operation will stop all the running appliance instances and remove them from the environment. After this operation is completed, the assembly instance will kept in the system so that it can be deployed again.
  - *RestartAssemblyInstance* This operation will restart all the running appliance instances of the assembly instance. The assembly instance will transition to the *Staged* and then transition back to *Deployed*.
  - *RedeployAssemblyInstance* This operation will redeploy the assembly instance. As part of this operation all appliance instances will be stopped and removed from the target environment. New appliance instances will be created and started.
  - *Scale* Scales the scaling group within an assembly instance. Scaling can be performed to scale up or down a scaling group with the assembly instance. The number of appliance instances that can be running for a scaling group must lie between its configured minimum and maximum instance limits. The Deployment continues to remain in the *Deployed* state.
- *Failed* When there is a failure in a deploy or undeploy operation, the assembly instances reaches this phase. A deployment operation may fail for a variety of reasons, such as insufficient resources. The operations that can be performed on a failed deployment are:
  - *DeleteAssemblyInstance* This operation will do the necessary cleanup (such as stopping and removing the appliance instances). After this operation is completed, the assembly instance no longer exists.
- *Staged* The staged phase is reached by stopping an assembly instance. In this phase all the appliance instances have been shut down. The operations that can be performed from this phase are:
  - *StartAssemblyInstance* This operation will start up all the appliance instances that have been shut down. After this operation is completed, the assembly instance is returned to the *Deployed* state.
  - *UndeployAssemblyInstance* This operation will remove all the appliance instances that have been shut down from the virtualized environment. After this operation is completed, the assembly instance will be kept around so that it can be deployed again.

1.3 Deployer Architecture

Oracle Virtual Assembly Builder Deployer is configured as a Web application in a WebLogic Server domain.

Figure 1–2 shows the top-level components of the Oracle Virtual Assembly Builder Deployer. The Deployer Repository contains the assembly archives and any deployment plans uploaded to the Deployer by the Web client. Assembly archives and deployment plans are created in Oracle Virtual Assembly Builder Studio, however, the repository maintained by the Deployer is separate from the Oracle Virtual Assembly Builder Studio catalog. Plans created in Oracle Virtual Assembly Builder Studio are uploaded to the Deployer through the Web service API. Runtime and configuration state for the Deployer is maintained in a distributed Coherence cache that is persisted so that single instance installations may recover in the case of a failure (that is, the Deployer process exits). The Deployer interacts with one or more virtualization systems and orchestrates the deployment of assembly archives into these systems.

Figure 1–2 Deployer Architecture



1.4 Targets

Different virtualization systems organize their resources in different ways and require different information for referencing and accessing them. In order to provide a common user experience across different systems, Oracle Virtual Assembly Builder Deployer defines the notion of a *target*. Targets are configured using administration interfaces defined later in this document and are used to reference a resource or pool of resources in the virtualized system. The configuration information provided for each target is specific to the virtualization system containing the target.

Oracle Virtual Assembly Builder Deployer provides support for deploying assembly instances to targets on Oracle VM.

1.5 Tags

Certain virtualization systems allow tags to be associated with artifacts such as templates and VM instances which may be queried by tools interacting with the system to locate artifacts or to find relationships between artifacts, for example finding the artifacts that are associated with a particular deployment.

In systems that support tagging, the Oracle Virtual Assembly Builder Deployer automatically adds tags to certain artifacts it creates. Table 1–1 provides information on these tags. The values for the tags are always strings, but certain ones may be interpreted as other types. The *Data Type* column indicates these types. For example, if the *Data Type* is Integer, then the value will be a string like "123", which can be interpreted as the number 123. The *Oracle VM Artifact* column indicates what kind of artifact or artifacts a particular tag is to be associated with.

Table 1–1 Oracle Virtual Assembly Builder Deployer Tags

Key Name	Data Type	Oracle VM Artifact	Description	Example
ovab:AssemblyArchiveName	String	Instance, template	The name of the assembly archive used to create the template that is the source of the instance	MySite.ova
ovab:TemplateId	String	Instance	The Oracle VM name of the template that is the source of the instance.	234jlk234jj3lkj43
ovab:TemplateRepositoryName	String	Instance	The name of the repository containing the template that is the source of the instance.	MyRepository
ovab:AppliancePath	String	Instance	The path to the appliance within the Oracle Virtual Assembly Builder assembly. This is used to determine the instance's place in the original assembly structure.	/MyWls/EjbCluster
ovab:ApplianceInstanceIndex	Integer	Instance	The appliance instance index.	2
ovab:ComponentType	String	Instance	The appliance type determined by Oracle Virtual Assembly Builder introspection.	WLS
ovab:DeployerId	String	Instance, assembly archive, template	An identifier for the Deployer instance that created the artifact.	
ovab:AssemblyInstanceId	String	Instance	An identifier for the assembly instance that is unique within the scope of the Deployer instance	Lkj234lk32j4lkj34
ovab:DeploymentPlan	String	Instance	The name of the deployment plan used for the deployment that the instance is a part of.	MyPlan
ovab:UserName	String	Template, assembly archive, instance	The name of the user that initiated the deployment operation that lead to the creation of the artifact.	John
ovab:InitialTargetCount	Integer	Instance	The initial target for the number of instances at deployment time.	5
ovab:ScalabilityMin	Integer	Instance	The minimum number of instances allowed for the appliance.	1
ovab:ScalabilityMax	Integer	Instance	The maximum number of instances allowed for the appliance.	10

1.6 Metadata-Driven Functionality

The details of the deployment orchestration are driven by OVF metadata which is embedded in the assembly archive that represents the application, and information stored in a deployment plan which you provide when creating a deployment.

2 API Web Services

The following sections describes the Oracle Virtual Assembly Builder Web Service API:

- Section 2.1, "Operations"
- Section 2.2, "Administrative Operations"
- Section 2.3, "Usage and Lifecycle"
- Section 2.4, "Admin Web Service"
- Section 2.5, "Deployer Web Service"

2.1 Operations

The Oracle Virtual Assembly Builder Deployer provides logical operations which are used in the Web service and CLI operations. Some operations are asynchronous and the status of these operations can be tracked with a `describeRequests` operation.

The complete API reference for the operations are described in the following chapters:

- Chapter 3, "API Reference: Administrative Operations"
- Chapter 4, "API Reference: Deployer Operations"

2.2 Administrative Operations

Oracle Virtual Assembly Builder Deployer provides logical administrative operations for configuring targets, which reference pools in virtualization environments. All of these operations are synchronous.

Due to the differences in how virtualization environments work and how we interact with them, configuration of these involves two steps.

- Creating targets using the `CreateTarget` operation. This step is required for Oracle VM
- Adding user configuration information for a named user to a target using `AddTargetUser`. For more details, see Section 3.3, "AddTargetUser" in Chapter 3, "API Reference: Administrative Operations".

Operations for adding users are not provided in these administrative operations. It is intended that these operations be performed directly against the LDAP server by security administrators authorized for such duties.

2.3 Usage and Lifecycle

The first operation when using the Oracle Virtual Assembly Builder Deployer is to upload an assembly archive to the Deployer's repository using the `uploadAssemblyArchive` operation.

You can register the uploaded assembly archive with one or more targets using the `registerAssemblyArchive` operation. The details of registration are specific to the target type. For Oracle VM, registration includes uploading the assembly archive to the Deployer and registering the templates with a target within the Oracle VM system.

After an assembly archive has been registered with a target, you can create one or more assembly instances for that registration using the `createAssemblyInstance` operation. An assembly instance is a context used to manage the lifecycle of VM instances for the assembly defined by the assembly archive. When creating an assembly instance, a deployment plan must be provided which defines the environment-specific details of the assembly managed through the assembly instance.

Once an assembly instance has been created for an assembly archive, you can initiate a deployment operation. During deployment, the initial instances for the assembly defined by the assembly archive are created and started. As this is done, the Deployer interacts with reconfiguration logic embedded in the initialization scripts of the instances which configures networks and mounted disk volumes and also configures the application stack for the environment. These interactions configure connections between instances that are created when the application stack starts up.

Once an assembly instance has been deployed, a number of other lifecycle operations can be applied to the running system such as `scale` which increases or decreases the number of running instances for an appliance within the assembly. There are also operations to stop and start all instances of the deployment, which are used for bringing the application offline and back online.

2.4 Admin Web Service

Operations against the Web service are made by posting an HTTP request to the Deployer’s context path. The request includes a request parameter that defines the action followed by zero or more request parameters that define arguments for the operation. See Section 2.5.2, "Query String Pattern" below for more details.

2.4.1 Context Path

The context path for the admin Web service is `/ovab/admin`.

2.4.2 Query String Pattern

Action= *action*&param1=*value*&param2=*value*...

For an array of values, use the same parameter name multiple times in the query string. For example:

Action= *action*&param1=*value*&param2=*value*&param2=*value*...

2.4.3 Actions

Chapter 3, "API Reference: Administrative Operations"describes all of the operations exposed by the admin Web service. Details of the types included in the "Response Content" column may be found in the schema in Appendix A, "Web Service Schema".

2.4.4 Error Handling

Table 2–1 shows how error handling is performed for the Web service.

Table 2–1 Web Service Error Handling

Responsible for Error	HTTP Status Code	Response Content XML Element
User	4xx	ErrorResult (type=USER)
Service	5xx	ErrorResult (type=SERVICE)

Note: In the case of an error, there may not be any content (`ErrorResult`) in the response. If the HTTP status code is generated by the servlet container, then there will not be any content in response. If the HTTP response is generated by the Oracle Virtual Assembly Builder Deployer, then most often there will be content in response, except when an unexpected service error occurs.

2.4.5 XML Schema

You can find the schema for the Web service in Appendix A, "Web Service Schema". This schema includes definitions for the response content items included in the table of actions, above.

2.5 Deployer Web Service

Operations against the Web service are made by posting an HTTP request to the Deployer's context path. The request includes a request parameter that defines the action followed by zero or more request parameters that define arguments for the operation. See Section 2.5.2, "Query String Pattern" below for more details.

As described in Section 2.1, "Operations", some operations may initiate an asynchronous action. The operations exposed by the Web service are the same as is described for the logical operations described earlier.

2.5.1 Context Path

The context path for the Deployer Web service is `/ovab/deployer`.

2.5.2 Query String Pattern

Action= *action*&param1=*value*&param2=*value*...

For an array of values, use the same parameter name multiple times in the query string. For example:

Action= *action*&param1=*value*&param2=*value*&param2=*value*...

2.5.3 Actions

Chapter 4, "API Reference: Deployer Operations" describes all of the operations exposed by the Deployer Web service. Details of the types included in the "Response Content" column may be found in the schema in Appendix A, "Web Service Schema".

2.5.4 Error Handling

Table 2–2 shows how error handling is performed for the Deployer Web service.

Table 2–2 Web Service Error Handling

Responsible for Error	HTTP Status Code	Response Content XML Element
User	4xx	ErrorResult (type=USER)
Service	5xx	ErrorResult (type=SERVICE)

Note: In the case of an error, there may not be any content (`ErrorResult`) in the response. If the HTTP status code is generated by the servlet container, then there will not be any content in response. If the HTTP response is generated by the Oracle Virtual Assembly Builder Deployer, then most often there will be content in response, except when an unexpected service error occurs.

2.5.5 XML Schema

You can find the schema for the Web service in Appendix A, "Web Service Schema". This schema includes definitions for the response content items included in the table of actions, above.

3 API Reference: Administrative Operations

The following sections describes administrative operations in the Oracle Virtual Assembly Builder Deployer Web Service API:

3.1 Parameters in HTTP Query String

The following designations apply to parameters in the "Other Parameters in HTTP Query String" sections of each operation:

- 1..1 : Parameter is required, and can have exactly one occurrence.
- 1..* : Parameter is required, and can have multiple occurrences.
- 0..1 : Parameter is optional, and can have exactly one occurrence, if provided.
- 0..* : Parameter is optional, and can have multiple occurrences, if provided.

3.2 Response Content

The response types for the operations correspond to definitions in the Web service schema. See Appendix A, "Web Service Schema".

3.3 AddTargetUser

This operation adds user configuration information for the named user to the target. The purpose of `AddTargetUser`, in addition to indicating that a user will be using the target, is to supply user specific information to be used when interacting with the target, if required.

For the Deployer which is used with Oracle VM, `AddTargetUser` is called by the Cloud Administrator to grant access to the target. In this case, no other properties are provided because the Cloud Administrator provides credentials for the virtualization environment when the target is created. Application Administrators are not allowed to call this operation for the generic Deployer because the credentials must be protected.

Other Parameters in HTTP Query String

target 1..1

user 1..1

Other parameters are used as properties for the user

HTTP Method

POST

Request Content

N/A

Response Content

text/xml Element: AddTargetUserResult

Required Role and Authorization

In the Oracle VM Deployer, only the Cloud Administrator is allowed to call `AddTargetUser`. The credentials are provided in the target configuration and must be protected.

A Cloud Administrator who calls this operation can specify a username other than their own.

3.4 CreateTarget

This operation creates a target for deployment operations. The call includes configuration information that defines the connection information. Credentials are supply for the Oracle VM target.

Other Parameters in HTTP Query String

name 1..1

type 1..1

default 0..1 (default value is false)

Other parameters are used as properties for the target

HTTP Method

POST

Request Content
N/A

Response Content
text/xml Element: CreateTargetResult

Required Role and Authorization
This operation can only be called by the Cloud Admin.

3.5 DeleteTarget

This operation deletes the deployment target and all configuration information. If this target was a default for a user or all users, then that default is unset.

Other Parameters in HTTP Query String
name 1..1

HTTP Method
POST

Request Content
N/A

Response Content
text/xml Element: DeleteTargetResult

Required Role and Authorization
This operation can only be called by the Cloud Admin.

3.6 DescribeTargetConfigurations

This operation describes configuration information for a target.

Other Parameters in HTTP Query String
target 1..1

HTTP Method
GET

Request Content
N/A

Response Content
text/xml Element: DescribeTargetConfigurationsResult

Required Role and Authorization
This operation can only be called by the Cloud Admin.

3.7 DescribeTargetNames

This operation lists created targets for deployment operations for the given type.

Other Parameters in HTTP Query String

type 1..1

HTTP Method

GET

Request Content

N/A

Response Content

text/xml Element: ListTargetsResult

Required Role and Authorization

This operation can be called by the Application Admin or Cloud Admin.

3.8 DescribeTargetUsers

This operation lists users for the given target.

Other Parameters in HTTP Query String

target 1..1

HTTP Method

GET

Request Content

N/A

Response Content

text/xml Element: DescribeTargetUsersResult

Required Role and Authorization

This operation can be called by the Application Admin or Cloud Admin.

3.9 DescribeUserTargets

This operation lists targets for the given user.

Other Parameters in HTTP Query String

user 1..1

HTTP Method

GET

Request Content

N/A

Response Content
text/xml Element: DescribeUserTargetsResult

Required Role and Authorization
This operation can be called by the Application Admin or Cloud Admin.

3.10 GetDefaultTarget

This operation gets the default target of that type for all users.

Other Parameters in HTTP Query String
type 1..1

HTTP Method
GET

Request Content
N/A

Response Content
text/xml Element: GetDefaultTargetResult

Required Role and Authorization
This operation can be called by the Application Admin or Cloud Admin.

3.11 GetTargetType

This operation gets the target type for the given target name.

Other Parameters in HTTP Query String
target 1..1

HTTP Method
GET

Request Content
N/A

Response Content
text/xml Element: GetTargetTypeResult

Required Role and Authorization
This operation can be called by the Application Admin or Cloud Admin.

3.12 RemoveTargetUsers

This operation removes a user from a target.

Other Parameters in HTTP Query String
target 1..1

user 1..*

HTTP Method

POST

Request Content

N/A

Response Content

text/xml Element: RemoveTargetUsersResult

Required Role and Authorization

This operation can be called by the Application Admin or Cloud Admin.

A caller holding the Cloud Admin role can call the method and specify a username other than their own. A non-admin user can only specify their own username.

3.13 SetDefaultTarget

This operation sets the target type of the given name as the default target for all targets of that type for all users. The specified target is used if the user calls an Oracle Virtual Assembly Builder operation and has neither specified a target nor has previously called `setUserDefaultTarget` for the target type.

Other Parameters in HTTP Query String

name 1..1

HTTP Method

POST

Request Content

N/A

Response Content

text/xml Element: SetDefaultTargetResult

Required Role and Authorization

This operation can only be called by the Cloud Admin.

4 API Reference: Deployer Operations

The following sections describes Deployer operations in the Oracle Virtual Assembly Builder Deployer Web Service API which are used to manage assembly archives and deployment aspects of the software system contained therein:

4.1 Parameters in HTTP Query String

The following designations apply to parameters in the "Other Parameters in HTTP Query String" sections of each operation:

- 1..1 : Parameter is required, and can have exactly one occurrence.
- 1..* : Parameter is required, and can have multiple occurrences.
- 0..1 : Parameter is optional, and can have exactly one occurrence, if provided.
- 0..* : Parameter is optional, and can have multiple occurrences, if provided.

4.2 AddAssemblyUsers

This operation grants access to an assembly to other users. This operation can only be called by the assembly owner or Cloud Admin. The owner is defined to be the user who first uploaded the assembly to the Deployer.

A user who is granted access may register the assembly with a target and create an assembly instance.

Other Parameters in HTTP Query String

user 1..*

assembly.name 1..1

HTTP Method

POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: AddAssemblyUsersResult

4.3 CreateAssemblyInstance

This operation creates an assembly instance for an assembly.

Other Parameters in HTTP Query String
assembly.name 1..1

assembly.version 1..1

target 1..1

plan.name 0..1

HTTP Method
POST

Request Context
application/octetstream

Plan file

Behavior
Synchronous.

Response Content
text/xml Element: CreateAssemblyInstanceResult

4.4 CreateTags

This operation creates one or more tags for a resource.

Other Parameters in HTTP Query String
tag 1..*

resource.id 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: CreateTagsResult

4.5 DeleteAssemblyArchive

This operation deletes an assembly archive from the Deployer. This operation may only be performed if there are no registrations for the assembly archive.

Other Parameters in HTTP Query String
assembly.name 1..1

assembly.version 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DeleteAssemblyArchiveResult

4.6 DeleteAssemblyInstance

This operation deletes an assembly instance. This operation can only be executed when the assembly is in an undeployed state.

Other Parameters in HTTP Query String
assembly.instance.id

1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DeleteAssemblyInstanceResult

4.7 DeleteAssemblyResources

This operation deletes resources associated with an assembly version.

Other Parameters in HTTP Query String

assembly.name

1..1

assembly.version

1..1

file

0..1 (a list of resource paths to be deleted. If unset, will delete all resources).

HTTP Method

POST

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DeleteAssemblyResourcesResult

4.8 DeleteDeploymentPlan

This operation deletes a deployment plan from the Deployer repository (note, this operation fails if you try to delete a plan that is in use).

Other Parameters in HTTP Query String

assembly.name

1..1

assembly.version

1..1

assembly.plan.name

1..1

HTTP Method

POST

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DeleteDeploymentPlanResult

4.9 DeleteLogEvents

This operation deletes log events in the system. If you do not supply any parameters, all events are deleted.

Other Parameters in HTTP Query String

event.type 0..1

user 0..1

assembly.instance.id 0..1

HTTP Method

POST

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DeleteLogEventsResult

4.10 DeleteRequests

This operation deletes one or more previously completed requests.

Other Parameters in HTTP Query String

request.id 1..1

HTTP Method

POST

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DeleteRequestsResult

4.11 DeleteTags

This operation deletes one or more tags for a resource.

Other Parameters in HTTP Query String

tag 1..*

resource.id 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DeleteTagsResult

4.12 DeployAssemblyInstance

This operation deploys an assembly instance. This operation orchestrates the creation of the initial instances for the deployment defined in the OVF and deployment plan.

Other Parameters in HTTP Query String
assembly.instance.id 1..1

HTTP Method
POST

Request Contexrequest contextt
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: DeployAssemblyInstanceResult

4.13 DescribeAssemblyArchives

This operation describes one or more assembly archives in the Deployer repository.

Other Parameters in HTTP Query String
assembly.name 1..*

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeAssemblyArchivesResult

4.14 DescribeApplianceInstances

This operation describes one or more deployed instances of an assembly.

In addition to information about the IDs and states of the deployed instances, the response content includes a list of elements that indicate the name, IP address, network name, and public/private status for each network interface of an appliance instance.

Other Parameters in HTTP Query String

assembly.instance.id 0..1

appliance.id 0..1

instance.id 0..1

HTTP Method

GET

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DescribeApplianceInstancesResult

4.15 DescribeAssemblyInstances

This operation describes one or more assembly instances.

Other Parameters in HTTP Query String

assembly.instance.id 1..*

HTTP Method

GET

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DescribeAssemblyInstancesResult

4.16 DescribeAssemblyResources

This operation describes resources associated with an assembly.

Other Parameters in HTTP Query String

assembly.name

1..1

assembly.version

1..1

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeAssemblyResourcesResult

4.17 DescribeAssemblyUsers

This operation describes one or more users of an assembly.

Other Parameters in HTTP Query String
assembly.name 1..1

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeAssemblyUsersResult

4.18 DescribeDeployer

This operation provides version information for the Deployer.

Other Parameters in HTTP Query String
N/A

HTTP Method
GET

Request Context
N/A

Behavior

Synchronous.

Response Content

text/xml Element: RedeployAssemblyInstanceResult

4.19 DescribeLogEvents

This operation describes log events in the system. If you do not supply any parameters, all events are described, otherwise, the result is filtered by the parameters you supply.

The Deployer tracks certain VM lifecycle events for viewing by an OVAB administrator. The events are:

- VM_CREATE
- VM_START
- VM_STOP
- VM DESTROY

The following information is stored for each VM_START, VM_STOP and VM_ DESTROY event:

- user
- timestamp
- deployment ID
- assembly name
- appliance path
- appliance instance index

The following information is stored for each VM_CREATE operation:

- CPU requirement
- MEMORY requirement
- DISK requirement

Other Parameters in HTTP Query String

user 0..1

assembly.instance.id 0..1

after.time 0..1

before.time 0..1

HTTP Method

GET

Request Context

N/A

Behavior

Synchronous.

Response Content
text/xml Element: DescribeLogEventsResult

4.20 DescribeRegistrations

This operation describes one or more assembly registrations.

Other Parameters in HTTP Query String
assembly.name 1..1

assembly.version 1..1

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeRegistrationsResult

4.21 DescribeRequests

This operation describes one or more previously issued requests.

Other Parameters in HTTP Query String
request.id 1..*

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeRequestsResult

4.22 DescribeScalingGroups

This operation describes one or more scaling groups.

Other Parameters in HTTP Query String
assembly.instance.id 0..*

scalinggroup.id 0..*

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeScalingGroupsResult

4.23 DescribeTags

This operation describes one or more tags for a resource.

Other Parameters in HTTP Query String
tag 1..*

resource.id 1..1

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeTagsResult

4.24 DescribeTargets

This operation describes one or more assembly instance targets.

Other Parameters in HTTP Query String
target 1..*

HTTP Method
GET

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: DescribeTargetsResult

4.25 DescribeVnets

This operation describes one or more networks.

Other Parameters in HTTP Query String

target 1..1

id 0..*

HTTP Method

GET

Request Context

N/A

Behavior

Synchronous.

Response Content

text/xml Element: DescribeVnetsResult

4.26 DownloadAssemblyArchive

This operation downloads an assembly from the Deployer.

Other Parameters in HTTP Query String

assembly.name 1..1

assembly.version 1..1

HTTP Method

POST

Request Context

N/A

Behavior

Synchronous.

Response Content

application/octet-stream

assembly archive (.ova file)

4.27 DownloadAssemblyMetadata

This operation downloads assembly metadata descriptor from the Deployer.

Other Parameters in HTTP Query String

assembly.name 1..1

assembly.version 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
application/octet-stream

assembly metadata (.ovf file)

4.28 DownloadAssemblyResources

This operation downloads a resources zip file from the Deployer repository.

Other Parameters in HTTP Query String
assembly.name

1..1

assembly.version

1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
application/octet-stream Assembly Resources Zip file

4.29 RedeployAssemblyInstance

This operation redeploys an assembly instance.

Other Parameters in HTTP Query String
assembly.instance.id

1..1

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: RedeployAssemblyInstanceResult

4.30 RegisterAssemblyArchive

This operation registers the archive with the backend virtualization system (Oracle VM).

Other Parameters in HTTP Query String
assembly.name

1..1

assembly.version

1..1

assembly.desc

1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: RegisterAssemblyArchiveResult

4.31 RemoveAssemblyUsers

This operation removes users from an assembly.

Other Parameters in HTTP Query String
user 1..*

assembly.name 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: RemoveAssemblyUsersResult

4.32 RestartAssemblyInstance

This operation restarts an assembly instance.

Other Parameters in HTTP Query String
assembly.instance.id 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: RestartAssemblyInstanceResult

4.33 ResumeAssemblyInstance

This operation resumes all vServers of an assembly instance.

Other Parameters in HTTP Query String
assembly.instance.id 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: ResumeAssemblyInstanceResult

4.34 ScaleAppliance

This operation scales an appliance. This operation changes the number of instances desired for an appliance and will lead to instances being created and started or stopped and destroyed depending on the new number.

Other Parameters in HTTP Query String
scalinggroup.id 1..1

target 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: ScaleApplianceResult

4.35 StartAssemblyInstance

This operation starts an assembly instance.

Other Parameters in HTTP Query String
deployment.id 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: StartAssemblyInstanceResult

4.36 StopAssemblyInstance

This operation stops an assembly instance.

Other Parameters in HTTP Query String
assembly.instance.id 1..1

force 0..1 (default is false)

HTTP Method
POST

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: StopAssemblyInstanceResult

4.37 SuspendAssemblyInstance

This operation suspends all vServers of an assembly instance. This is a virtual hibernate operation where the OS is not shut down.

Other Parameters in HTTP Query String

assembly.instance.id 1..1

HTTP Method

POST

Request Context

N/A

Behavior

Asynchronous.

Response Content

text/xml Element: SuspendAssemblyInstanceResult

4.38 UndeployAssemblyInstance

This operation undeploys the assembly instance.

Other Parameters in HTTP Query String

assembly.instance.id 1..1

HTTP Method

POST

Request Context

N/A

Behavior

Asynchronous.

Response Content

text/xml Element: UndeployAssemblyInstanceResult

4.39 UnregisterAssemblyArchive

This operation removes the registration artifacts from the backend virtualization system (target). This operation can only be called if there are no assembly instances for this assembly archive in the target.

Other Parameters in HTTP Query String

assembly.version 1..1

assembly.name 1..1

target 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: UnregisterAssemblyArchiveResult

4.40 UpdateAssemblyArchive

This operation allows you to modify the description information for the assembly archive.

> **Note:** To upload a new version, you would call `UploadAssemblyArchive` again.

Other Parameters in HTTP Query String
assembly.name 1..1

assembly.version 1..1

assembly.desc 1..1

HTTP Method
POST

Request Context
N/A

Behavior
Synchronous.

Response Content
text/xml Element: UpdateAssemblyArchiveResult

4.41 UploadAssemblyArchive

This operation is used to upload an assembly operation from a client to the Deployer. The Deployer will keep multiple versions of an assembly archive so you may call this operation more than once for the same assembly.

Other Parameters in HTTP Query String
assembly.name 1..1

HTTP Method
POST

Request Context

application/octetstream

Assembly archive (.ova file)

Behavior

Synchronous.

Response Content

text/xml Element: UploadAssemblyArchiveResult

4.42 UploadAssemblyResources

This operation uploads a resources zip file to the Deployer repository, and associates it with an assembly version.

You typically handle assembly resources as a set and would upload the resources in a single operation. If you needed to replace the resources you delete all of the resources and replace them with an upload of a new set of resources. This is the default behavior of these operations.

In the case of disk images, you may prefer replace a single image without incurring the cost of re-uploading all the resources. Also, since scripts are generally not tightly coupled to the assembly itself, you may want to modify or add scripts without having to re-upload all the resources.

For these reasons, these operations have optional options to support these scenarios. The `DeleteAssemblyResources` operation has a `files` option that allow you to specify a subset of files to be deleted. The default behavior if this is not supplied is to delete all resources. You can use the `files` option in conjunction with the `append` option of `UploadAssemblyResources`. The `append` option provides a way to add to the existing set of resources. You can achieve the notion of "replace" by deleting a subset of the resources – using the `files` option – and then uploading replacements for just those files with the `append` option of `UploadAssemblyResources`.

Other Parameters in HTTP Query String

assembly.name

1..1

assembly.version

1..1

append

0..1 (Allows upload of additional resources. Without this option, an attempt to do a second upload is rejected).

HTTP Method

POST

Request Context

application/octet-stream Assembly Resources zip file

Behavior

Synchronous.

Response Content

text/xml Element: UploadAssemblyResourcesResult

4.43 UploadDeploymentPlan

This operation uploads a deployment plan for an existing assembly to the Deployer repository. Provides a mechanism to store deployment plans in the deployer repository associated with a specific assembly version. Because you can specify a plan name to the `CreateAssemblyInstance` operation that references a previously uploaded deployment plan, uploading a plan at assembly creation time is optional.

Other Parameters in HTTP Query String

assembly.instance.id

1..1

HTTP Method

POST

Request Context

application/octet-stream deployment plan xml or zip stream

Behavior

Synchronous.

Response Content

text/xml Element: UploadDeploymentPlanResult

4.44 ValidateAssemblyInstanceResources

This operation checks to see if the target of an assembly instance has enough resources to deploy the assembly. It checks the following resources:

- Memory availability
- Disk availability
- Network availability. Checks if the network with the name specified in the plan exists in the target.

`ValidateAssemblyInstanceResources` performs a deployer side check. There is also a `ValidateAssemblyArchiveResources` CLI operation which performs the same checks, but against an assembly archive that only exists on the client. There is no Web service equivalent of this operation. `ValidateAssemblyArchiveResources` works by querying the Deployer for current resource availability and then does the validation locally on the client using the data returned.

Other Parameters in HTTP Query String

assembly.instance.id 1..1

HTTP Method

GET

Request Context
N/A

Behavior
Asynchronous.

Response Content
text/xml Element: ValidateAssemblyInstanceResult

5 Preparing to Develop using the Oracle Virtual Assembly Builder Plug-in SDK

The following sections describes how

5.1 Introduction

Developers can use Oracle Virtual Assembly Builder to create an introspector plug-in.

5.1.1 What Does a Plug-in Do?

The process of capturing the state of the reference host begins with introspection. Your plug-in participates in introspection by performing dehydration of the reference software installation. The output of dehydration is metadata that describes the reference installation.

Later instantiation of the software appliance as a virtual machine in the target environment is known as deployment. Your plug-in participates in deployment by performing rehydration of the reference software within the context of the running virtual machine. Metadata recorded at dehydration time is available to the plug-in during rehydration.

5.1.2 Packaging and Distribution

Your plug-in is implemented as a .jar file that is packaged with the product you support. Your plug-in enters the Oracle Virtual Assembly Builder environment at runtime through the dynamic plug-in discovery process. After discovery, the Oracle Virtual Assembly Builder product framework arranges to invoke your plug-in's dehydration and rehydration entry points at the appropriate time and place: on the reference host during introspection for dehydration and within the running virtual machine late in deployment for rehydration.

5.2 Preparing to Develop a Plug-in

As you begin to design and implement your introspector plug-in these are the first things you should do.

5.2.1 Name the Plug-in

Choose a name for your introspector. Introspector plug-in jar files live within the Oracle Virtual Assembly Builder software installation directory tree, each plug-in in its own subdirectory. These subdirectories are named according to the plug-in name, so each plug-in name must be unique. Choose a name related to the product that will be introspected - this will make it easier for users of your plug-in to figure out how to invoke it. The name of the plug-in is used to create the CLI command name for introspecting your product. A command name of the format introspect<plug-in name> is constructed. In the GUI your plug-in name will show up "as is" in a list of component types that can be introspected.

5.2.2 Plug-in or Plug-in Extension

Decide if you will be writing a plug-in or an extension to an already existing plug-in. If your product runs on top of another product for which an Oracle Virtual Assembly Builder plug-in already exists (such as WLS), writing an extension to that plug-in may make sense.

5.2.3 Version the Plug-in

Devise a plan for dealing with the introspection of different versions of your product. This will have an impact on the name you pick for your introspector plug-in. There are two distinct strategies, one of which may work better for you than the other. One strategy is to handle the introspection of different product versions entirely within a single plug-in; this may be convenient if the product versions being supported have similar configuration data. For a product called "foo" you could name your introspector plug-in "foo", and the plug-in would detect the product version (version 1 or version 2) and do the appropriate thing for that version.

Alternatively, if the product versions are so different that it's more convenient to keep the introspector code completely separate, you could write two plug-ins named "foo1" and "foo2", each of which is capable of introspecting only a single product version. In this case a customer who had both product versions installed somewhere on their network might have both introspector plug-ins present in a single Oracle Virtual Assembly Builder instance, and the customer would have to know what product version they want to introspect. Note that due to Oracle Virtual Assembly Builder's classloading scheme you must use different package names in the different versions of your plug-in.

5.2.4 Developing an Introspection Plug-in

As you begin to write code for your introspector plug-in, see Chapter 6, "Developing an Introspection Plug-in" and Chapter 7, "Introspection Plug-in Module Reference". These chapters contain tutorial code samples, explain common plug-in operations, and provide a reference to the modules in the Oracle Virtual Assembly Builder introspection framework. You could read through the tutorials and copy the sample code to produce a plug-in that can be invoked by Oracle Virtual Assembly Builder, to which you can add your product-specific code.

5.3 Plug-in Architecture and Workflow

Introspector plug-ins are organized by the Introspector Manager. It is the core of the Introspector framework and acts as the controller for all plug-ins. The Introspector Manager is the layer that sits between the user interfaces (the Studio GUI and the command line utility `abctl`) and the plug-ins. It tracks all the plug-ins and invokes the plug-ins' dehydration and rehydration methods.

Since there can be a variable set of plug-ins, a mechanism is needed to inform the Introspector Manager of the exact set of plug-ins that are available. This is done using a service model provided by Oracle Virtual Assembly Builder called SPIF. To make the Introspector Manager aware of a new plug-in it need only be registered as an IntrospectorPlugin service. The details of how this registration is accomplished are described in Chapter 6, "Developing an Introspection Plug-in". Once a plug-in is registered the rest is handled by the Introspector Framework.

5.3.1 Workflow

The process of using Oracle Virtual Assembly Builder, and triggering interactions with the plug-in, is as follows:

1. The OVAB user begins an introspection, indicating the type of product component to be introspected, the host where that component resides, and other information necessary to obtain the product configuration data - such as passwords, directory paths, or user names.
2. The appropriate IntrospectorPluginLocator for the plug-in is located and is used determine if a newer version of the plug-in exists on the reference host. If so, the user is required to upgrade before introspection can be done.
3. If no newer version of the plug-in is found then the plug-in's `Dehydrator.dehydrate()` method is invoked, executing on either the local or a remote host, as specified by the user. The dehydrator captures the key configuration information and creates Oracle Virtual Assembly Builder metadata. It indicates what product executables, data and configuration need to be copied from the introspected system, describing them in terms of file set definitions.
4. The Oracle Virtual Assembly Builder framework captures file sets to be created from the introspector plug-in's file set definitions - by default this is done immediately after introspection completes, but at the user's option it may be deferred until a later time.
5. The OVAB user creates an assembly from various introspected components (represented in the metadata as appliances or atomic assemblies), describing their relationship to each other in terms of network connections. The user may edit other configuration parameters at this time, as well.
6. The OVAB user creates a deployable VM template, which combines the file sets for the appliances in the assembly with a VM system base image. These templates, along with the assembly metadata and the Oracle Virtual Assembly Builder bits necessary for deployment (including the introspector plug-ins) are then combined into an assembly archive (.ova file).
7. The OVAB user creates a deployment plan. The deployment plan can be used to override any editable metadata. It is also used to specify network configuration for the target OVM environment that will be used for deployment.

8. At deployment time, a VM is instantiated for each appliance, and the introspector plug-in's `Rehydrator.rehydrate()` method is run. The plug-in performs the component rehydration by converting the saved metadata state into a product configuration appropriate for the VM environment, with a modified network configuration, a new host name, etc. After the `rehydrate()` method completes, the `Rehydrator.start()` method is called. The `start()` method performs the necessary actions to start the component.
9. After the assembly is deployed it can be stopped and started as a whole, and scalable appliances can be scaled up or down.

5.3.2 How Does Your Plug-in Fit in

An introspector plug-in participates in both the capturing (dehydration) of a product from a reference host as well as the deployment of that product as a virtual appliance (rehydration). Each plug-in must inspect the configuration of the product on a reference host and capture enough of the data to be able to deploy the product on a virtual system. During rehydration the product configuration must be fixed to make the product work on the new host, and to apply any configuration overrides the user has specified.

5.3.2.1 Job Parameters

Your plug-in can specify parameters to be supplied by the user when they introspect your component. These parameters appear as command line options when using abctl, or as text field entries in the GUI. For details on the different job parameter types and how they are presented to the user read the Job Parameters tutorial in Chapter 6, "Developing an Introspection Plug-in".

Note that password type job parameters are treated differently from other parameter types by abctl for security reasons. Instead of the user entering the password so it appears in clear text on the command line, the user will be prompted to enter the password when the plug-in or extension asks for the value.

5.3.2.2 Dehydration

Dehydration is the process of capturing the relevant configuration information for a given product into metadata and a file set definition. The definition of 'relevant' is product-specific - there is no need to capture every possible configuration element so that the user can reconfigure the system completely from the introspected reference system to the deployed virtual system, because reconfiguration is the domain of Oracle Enterprise Manager. The dehydration process must capture only enough information for the critical portions of a configuration that you expect a user would change from one system to another. For example, if your product has a password, it would be good if the user were allowed to change the password prior to deploying a new system.

Note that the product's installation paths specified in the file set definition remain the same on the deployed VM as they were on the reference host, so your plug-in doesn't need to do any path fix up and should not expose file set paths to the user for customization.

Your plug-in must provide a Dehydrator implementation. The `Dehydrator.dehydrate(DehydrateContext context)` method is called during introspection. The `DehydrateContext` passed in to this method gives you access to the JobParameter values supplied by the user. It also has methods used to create the Appliance or Assembly that will be the result of dehydration.

There are different types of metadata that can be used to describe different aspects of your product and these will be described in detail in Section 5.4, "Metadata Overview".

5.3.2.3 Rehydration

Rehydration is the opposite of dehydration. Rehydration is the process of taking captured metadata and applying it to a virtual system. Rehydration occurs on the deployed virtual server and is invoked by the Deployer as the last step of deployment.

Your plug-in must provide a Rehydrator implementation. The `Rehydrator.rehydrate(RehydrateContext context)` method is called when it is time for your plug-in to reconfigure your product during deployment. The `RehydrateContext` passed in to your `rehydrate()` method provides access to the metadata associated with the appliance being deployed.

The following methods may be of particular interest:

- `RehydrateContext.getTargetAppliance()` gets the target appliance (the appliance being deployed) and access properties, inputs, outputs, etc.
- `RehydrateContext.getHostForTargetAppliance()` gets the VM hostname

There are some things you should know about the virtual machine environment during rehydration:

- The Rehydrate job always runs as 'root' on the deployed system. Any files you copy or create will have root ownership, etc. You will need to change the user and group ownership of any files you create during rehydration as appropriate.
- The files included in a file set definition will automatically get set to the 'owner' and 'group' values in the definition. If no value is specified by the plug-in during dehydration or by the user when the assembly is edited, both the owner and group will be set to 'oracle'.
- The files included in file set definitions will be created with their original permissions on the VM.
- RehydrateUtils in the oracle.as.assemblybuilder.introspector.plugin.util module has the ability to run commands as a specific user. Make sure to start processes as 'oracle' where applicable.
- Your plug-in code has access to all the metadata in the assembly during rehydration. You can use the RehydrateContext.getAssemblyNavigator() method to get an AssemblyNavigator to help in finding the metadata you need.

5.3.2.4 Post-Deployment Operations

Introspector plug-in Rehydrators are involved in other operations in addition to rehydration. Deployed assemblies can be stopped and started . The Rehydrator.stop() method gives you a chance to shut down your component gracefully when the deployed assembly is stopped. The `Rehydrator.start()` method is called during deployment right after the `Rehydrator.rehydrate()` method is called, and it will also be invoked when an assembly is started after being stopped.

5.4 Metadata Overview

These sections describe the various Oracle Virtual Assembly Builder metadata constructs that the introspector plug-in writer should be familiar with. The Oracle Virtual Assembly Builder metadata is mostly stored as XML files in the Oracle Virtual Assembly Builder Catalog, under the $AB_INSTANCE/catalog directory.

5.4.1 Type of Metadata

Before you continue through the metadata overview, consider what kind of metadata your plug-in should create as a result of introspecting your product.

5.4.1.1 Appliance

The simplest option is to produce a single appliance, which is metadata and associated product configuration data that may eventually be deployed as a single Virtual Machine.

5.4.1.2 Atomic Assembly

A more complex option is to make your plug-in create an atomic assembly, which is a collection of related appliances whose shape cannot be altered by the user - that is, appliances cannot be added to or removed from the atomic assembly, and any connections between those appliances cannot be changed during assembly editing. Each appliance in an atomic assembly will be instantiated as its own VM.

5.4.2 Appliances

An *appliance* is the primary output of the introspection process. An appliance is a metadata construct that can (after additional configuration steps) be instantiated as a running VM. Part of the appliance metadata describes the configuration of the product environment on the introspected system. An appliance can also have associated *content resources*, such as product configuration and other files, that are necessary to instantiate the product on a VM. This product data and appliance metadata is stored in the Oracle Virtual Assembly Builder catalog.

The appliance also has a set of resource requirements, which for instance allow the plug-in writer to specify that a VM instantiating the appliance should have a minimum amount of memory or a minimum number of CPUs available to it.

Each appliance also has scalability information that describes the number of VMs that should be created from the appliance - minimum, target, maximum, and absolute maximum. The "target" value is the initial number of VM instances of the Appliance that will be created by default. The minimum and maximum values are bounds on the number of VM instances, and these bounds are in effect after the Assembly containing the Appliance is deployed. The "absolute maximum" value is an upper limit on the maximum value. Oracle Virtual Assembly Builder does not support an absolute maximum of more than 2000.

An appliance also has *interfaces*, which describe its relationship to the network configuration on the VM's host, as well as *inputs* and *outputs*, which describe network connections to other appliances.

The appliance *creator* is the introspector plug-in or extension name, and cannot be altered by plug-ins. It is added to the appliance automatically. This value is used by the Oracle Virtual Assembly Builder framework to determine what plug-in to use to operate on the appliance.

The appliance *type* defaults to the plug-in or extension name, but a different value can be specified by the plug-in if it is desired. It generally only makes sense to specify a type different from the default if the same plug-in can create multiple appliance types.

5.4.2.1 Scalability

After an assembly is deployed, the user may choose to scale the appliances, which is to say that the user may create additional VMs that instantiate the same appliance. The introspector plug-in can specify restrictions on the scaling behavior of the appliances it creates, by specifying minimum and maximum numbers of instantiations of each appliance. Decide whether it makes sense for your appliance(s) to be scalable, and if so what the bounds should be.

5.4.3 Assemblies

An assembly is a container for appliances and sub-assemblies, allowing us to maintain connections between them. Only a non-atomic assembly can have a deployment plan, so a lone appliance or atomic assembly not contained in a 'top-level' assembly cannot be deployed to the virtual environment. (The simplest assembly contains only a single appliance, and the deployment of that assembly would result in a single appliance instance, or VM.)

5.4.4 Atomic Assemblies

Assemblies created by introspector plug-ins are called *atomic assemblies*. Atomic assemblies may not contain sub-assemblies. Oracle Virtual Assembly Builder Studio restricts the editing operations that users can effect on atomic assemblies - the connections between appliances created by the plug-in in an atomic assembly cannot be edited by the user, nor can appliances be added to or deleted from an atomic assembly.

An atomic assembly is deployed as a unit, so the appliances created in the assembly by the plug-in should have their network connections in place where appropriate.

All appliances in an atomic assembly must share file sets. This is because the introspection process only goes to one host to capture the file sets. Appliances that share file sets are called *siblings*; after the first appliance is created, siblings are added to the atomic assembly using the `Assembly.createSibling()` method.

For example, in Oracle WebLogic Server, the Admin Server contains all the same files as any of the Managed Servers, so the Admin Server is the only machine that needs its file sets captured, and the other appliances can be siblings of the Admin Server appliance.

If, however, your product consists of multiple types of different appliances that need to have file sets captured separately, then you need to have your plug-in generate appliances and not an atomic assembly. You may need multiple plug-ins, or you may be able to accomplish this using a single plug-in. Each appliance may then have its file sets captured separately and then it will be left to the User to assemble these appliances into a proper assembly with all the necessary connections.

5.4.5 Properties

A property is a name-value pair that stores values of various types (string, integer, password, boolean and file reference). Most other Oracle Virtual Assembly Builder metadata constructs can have properties attached to them, including assemblies, appliances, inputs, outputs and interfaces.

5.4.5.1 Property Categories

There are two categories of properties: user and system, which differ in that user properties can be overridden by the OVAB user whereas system properties are read-only. Either type of property can be marked as required or not, and in the case of a user property, an initial value can be specified, or not. Before an assembly can be deployed, values must be specified for all required properties on all metadata associated with that assembly. These values can be specified in the assembly metadata during assembly editing or via the deployment plan.

5.4.5.2 Property Groups

Properties can be sorted into property groups by attaching a common prefix to the names of all the properties that are meant to be grouped together. Oracle Virtual Assembly Builder Studio uses groups as a way to organize the properties when they are displayed to the user. In plug-in code, groups of properties can be retrieved by specifying the name prefix associated with the group.

Properties not in defined groups are displayed in Oracle Virtual Assembly Builder Studio according to a heuristic that groups properties based on common prefixes (if any) and splits the property name string on the '.' character.

Note that property group names starting with "OVAB", "Ovab" or "ovab" are reserved for internal use.

5.4.5.3 Synthetic Properties

Synthetic properties are properties that do not exist in appliance metadata and are not created by plug-ins; they exist only in the deployment plan. For example, there is a synthetic property named com.oracle.ovab.deployer.anti-affinity-min-servers that may be overridden in a deployment plan to specify how many physical machines should be used to host the VMs for an appliance. This property is only relevant for instantiated appliances and hence does not exist in the appliance metadata.

5.4.5.4 Setting and Retrieving Properties

For most property types, creating the property during dehydration and retrieving its value during rehydration is straightforward. Password properties and file reference properties need some additional explanation though.

5.4.5.4.1 Passwords Passwords are encrypted when the metadata is persisted to disk in the Oracle Virtual Assembly Builder catalog. When retrieving the password value during rehydration you need to use the `TypedValue.getPasswordValue().getAsChar()` method to get the clear text value. You will get a `char[]` which, for security reasons, you should clear out as soon as you are done with the password, as follows: `Arrays.fill(passwd, (char) 0);`

5.4.5.4.2 File References A file reference property has a value that points to a file. The file can be provided by the plug-in during dehydration by specifying an existing content resource. A null value can also be specified. If the property is marked required and a null value is specified by the plug-in, the file must be provided by the user via the deployment plan.

To override a file reference type property via the deployment plan the user specifies a local file from disk. This file is then sent to the VM during deployment and can be accessed from the plug-in rehydration code.

Note: File reference properties are only visible in the deployment plan - they cannot be seen or edited during assembly editing.

To access the file during rehydration the plug-in must get the value of the FileRef property and parse it to find the name of a content resource. The property value will be a URI value like "contentresource:<name>". The methods java.net.URI.getScheme() and java.net.URI.getSchemeSpecificPart() may be useful for parsing the value. Once the content resource name is obtained the content resource can be accessed using the usual SDK methods provided for accessing content resources.

Oracle Virtual Assembly Builder reserves the right to support other URI schemes for FileRef property types in the future. Plug-in code should be robust to this, that is, check the URI scheme to ensure that it is "contentresource", and ignoring properties with an unknown scheme, before trying to look up a corresponding content resource. This is especially true if a plug-in is scanning through all available properties.

5.4.6 Inputs and Outputs

Inputs and outputs are features of appliances and atomic assemblies that describe network endpoints used by the appliance or atomic assembly. (The data structures are a bit different for appliances and assemblies, so you'll see classes for ApplianceInput, AssemblyInput, ApplianceOutput, and AssemblyOutput.)

Inputs and outputs are created by the introspector plug-in to indicate possible attachment points for connections, either to other endpoints in the same assembly, or to external resources. For instance, a database appliance has an input describing the port where the database listens for connections, and a WLS appliance has a JDBC output describing the host that is used for back-end storage. The WLS output can be connected to the database input. It is via this connection that the database host and port can be known to the WLS plug-in at deployment time, so it can fix its JDBC configuration.

When viewing appliances in the Oracle Virtual Assembly Builder Studio assembly editor, inputs appear as bisected circles on the left side of the appliances, whereas outputs appear as bisected circles on the right side.

Inputs have an associated port number, which is the network port number where they accept network connections, and a list of network protocols that are supported on that port. Inputs are bound to one of the Appliance's network interfaces.

Both inputs and outputs can have properties attached to them by the introspector plug-in.

Outputs have a single network protocol associated with them, the one that is used for communication from that output.

5.4.6.1 Output Connection Properties

Outputs can have an optional associated set of connection properties, which (if present) are used to ensure that the output is connected to a compatible input - if the names of all output connection properties match names of input properties, the input and output are deemed to be compatible. The connection properties are also used to seed the input properties of an external resource created for that output.

Connection properties and input properties are a part of the external surface of the plug-in-generated metadata; they must be documented so that other components understand how to connect to your appliances and assemblies.

5.4.6.2 Connections

A connection represents the network connection from an output to an input. The two endpoints included in the connection can belong either to appliances or assemblies. When a connection is to or from an assembly, the endpoint is actually owned by an appliance in that assembly.

5.4.6.3 External Resources

In cases where an output refers to a network resource that is not represented by an appliance in the assembly, the user will be able to connect that output by creating an external resource in the assembly - the unbound appliance output will be configured to connect to an input on the new external resource. An external resource is a sort of placeholder, in that no VM is created for an external resource when the assembly is instantiated. They exist in the assembly solely to allow the user to specify configuration information (hostname, port, etc.) for pre-existing entities on the user's network.

For example many Oracle products depend on an LDAP server. Currently Oracle Virtual Assembly Builder does not support deploying an LDAP appliance into the virtual environment, so an external resource is used in the assembly to represent the LDAP server.

5.4.6.4 Vnets

When a non-atomic assembly is created, it is given a default Vnet, and additional Vnets can be added to it later. When the assembly is instantiated, these Vnets are associated with networks from the OVM environment. Appliances are created with interfaces that must be associated with an assembly Vnet before the assembly can be instantiated.

Manipulation of Vnets is not done by introspection plug-ins. Instead, users create and configure Vnets while composing a non-atomic assembly.

5.4.6.5 Interfaces

Interfaces in the metadata represent network interface cards (NICs). There are two types of interfaces. A physical interface represents a physical network interface card. Physical interfaces can then have associated virtual interfaces which are basically additional IP addresses on that same physical NIC.

There must be at least one interface on an appliance. If the plug-in does not create an interface on an appliance the introspection framework will automatically add a default one.

5.4.7 Content Resources

Content resources are files that are saved in the Oracle Virtual Assembly Builder catalog during dehydration for retrieval and use by the plug-in during rehydration. Content resources should not be used to transfer unmodified files from the introspected system to the deployed system - that is what file set definitions are for.

Content resources are meant to be used for files that you wish to turn into Velocity templates (or something similar) during dehydration for later token replacement during rehydration - without modifying the original file on the introspected system.

Content resources can also be used for files that you want the user to be able to update as a whole at assembly editing time. For instance so users could provide custom SSL certificates. In this case you must expose the content resource as a FileRef user property. Doing so allows the user to add a different copy of that file at deployment time.

Content resources can be associated with an appliance or an atomic assembly.

5.4.8 File Set Definitions

(These were previously known as Package Definitions and are created via the PackageDefinitionBuilder class.) In the simple case, File Set Definitions are essentially lists of files that are to be copied from the introspected system to the catalog, and later built in to the templates that will be used to instantiate the appliances.

Typically, the files listed in the definition are compressed and zipped (captured) into file sets (previously "packages") by the Oracle Virtual Assembly Builder framework for copying from the introspected system to the catalog. The file sets are unzipped onto VM disk images during the template creation process, and the disk images are mounted as directories on the running VMs as appliances are instantiated.

There are two types of file set definitions - user and system. User definitions can be edited by the user including being completely removed. System definitions cannot be edited other than selecting the 'type' as shared or local. System definitions are meant to indicate a set of files that are necessary for the product to function properly.

5.4.8.1 Exclusions

When creating your file set definition you can specify excludes. These are files that exist somewhere under the specified root directory that you do not want included when the file set is captured. Typical candidates for exclusion would be log files, which it does not make sense to carry over onto the deployed VM.

The exclude semantics are described in Table 5–1:

Table 5–1 Exclude Semantics

Directory Relative to the Root of the Definition	Effect of Exclusion
some/relative/path/foo	Excludes the entire directory 'foo'. The 'foo' directory **will not** exist at all on the deployed VM.
some/relative/path/bar/*	Excludes all the files under directory 'bar'. The empty directory **will** exist on the deployed VM.
some/relative/path/logs/*.log	Excludes all files in the logs dir with names ending in '.log'. Any other file is still captured. The logs directory **will** exist even if all files were removed from it.

5.4.8.2 Shared File Sets

The more complex scenarios supported by file set definitions involve shared file sets. In these cases various arrangements are made for a volume to be shared after VMs are instantiated. File set capture (known in the APIs as "packaging") is optional for shared file sets; when no file set is created, the file set definition may refer to a pre-existing volume or an empty volume.

When shared file sets are captured by the Oracle Virtual Assembly Builder framework, the file system type may be either NFS or DISK_IMAGE. In the NFS case, Oracle Virtual Assembly Builder will arrange for the contents of a temporary disk image to be copied on to an NFS file system that can be accessed by all the appliances. Subsequent appliance instances just mount the NFS file system. Note that NFS file sets that are captured can only be used between members of a clustered appliance. In the case of a captured DISK_IMAGE file set each appliance will see the contents of the file set mounted as a read-only disk. This is also only available to members of a clustered appliance.

If file sets are not being captured and added to the template by the Oracle Virtual Assembly Builder framework, a pre-existing shared volume is assumed, and the file system type should be either NFS or RAW.

Here is a summary of the shared file set possibilities:

Table 5–2 Shared File Sets (Captured)

File System Type	Sharing Rule
NFS	Can only be shared by members of a clustered appliance. An NFS volume external to the VM and specified by 'mount-target' in the deployment plan is created, populated and mounted when the first appliance in a cluster is instantiated. Each subsequent instantiation of a cluster member will simply NFS-mount this volume.
DISK_IMAGE	Can only be shared by members of a clustered appliance. Oracle Virtual Assembly Builder constructs a volume containing the specified files. The volume is included in each appliance's configuration by OVM. These volumes are read-only.
RAW	Not available.

Table 5–3 Shared File Sets (Not Captured)

File System Type	Sharing Rule
NFS	A preexisting NFS volume, specified by 'mount-target' in the deployment plan, is mounted by each appliance.
DISK_IMAGE	DO NOT USE: This will result in a read-only volume of the size specified that will remain forever empty. You do not want your plug-in to create one of these.
RAW	Can only be shared by members of a clustered appliance. Oracle Virtual Assembly Builder configures Linux raw devices and /dev for owner, group, permissions, and symbolic name. (Note: this exists to support the RAC database appliance, and most likely will not be used by any other plug-in type.)

Each file set definition includes a root directory, which is scanned recursively to collect files as file sets are captured. Files under the root directory may be excluded based on file names or patterns, in case particular files such as logs or temporary files do no need to be copied from the introspected system to the instantiated VM.

Atomic assemblies are restricted in their configurations in that all appliances in an atomic assembly share the same file sets. (Content resources, described previously, are specific to each appliance in the atomic assembly.)

5.5 Introspector Plug-in Extensions

A plug-in extension can be used to add dehydration and rehydration operations to an existing plug-in. This is a useful capability when one product is built on top of another. Rather than duplicating the work of the underlying product, a plug-in extension can be added to run when the underlying plug-in runs and contribute additional metadata and logic when appropriate.

5.5.1 Semantics of Plug-in Extensions

Plug-in extensions follow these semantics:

- A plug-in extension is always executed when its parent plug-in is executed.

- A plug-in extension can contribute to the set of parameters accepted by a plug-in during dehydration job submissions. These additional parameters are presented to plug-in extensions along with the parameters specified by the plug-in.
- A plug-in extension can choose to be represented as an "alias" of the parent plug-in for launching dehydration operations. The effect of this is for it to appear as if a separate additional introspection choice is available to the user though this does not have an effect on execution as the underlying plug-in is always executed along with all of the plug-in extensions that extend it.
- A plug-in extension can extend a plug-in or another plug-in extension (the entity from which a plug-in extension extends is considered the parent).
- A plug-in extension always executes after its parent executes.
- A plug-in extension can have only one direct parent.
- A plug-in or plug-in extension can have multiple children plug-in extensions.
- "Sibling" plug-in extensions execute in an arbitrary order.

5.5.2 Restrictions on Plug-in Extensions

Plug-in extensions follow these restrictions:

- Plug-in extensions are usually dependent on the work performed by its parents. A plug-in extension often needs to know what its parent contributes in order to use the data provided by the parent and not create any conflicting data. This implies that a plug-in extension may require modification when a parent is modified.
- Plug-ins must remain independent of the plug-in extensions that extend from it. A plug-in is not provided access to the plug-in extensions that extend from it and a plug-in extension should take care to do no harm to the operation of its parents.
- Plug-in extensions cannot create assemblies or appliances unless the parent plug-in permits it. Without agreement from the plug-in owner and careful management, the creation of appliance or assemblies by a plug-in extension could interfere with subsequent operations performed by the plug-in or other plug-in extensions. Without such an agreement, plug-in extensions must restrict their operations to only those that contribute to the metadata of assemblies and appliances created by the plug-in.
- A plug-in extension must do nothing when there is nothing for it to do - that is, when the component the extension is handling is not installed. Plug-in extensions always execute when the parent plug-in executes which implies that some executions will not require any work from a particular plug-in extension. It is the responsibility of the plug-in extension to anticipate this and not report any errors when there is nothing to do. This is true during both dehydration and rehydration.

5.5.3 Plug-in Extension Execution

At execution of dehydration, the plug-in is executed first with the output consisting of an Assembly/Appliance. Upon completion of the plug-in, each plug-in extension in turn is passed both the entire set of user specified job parameters and the Assembly/Appliance including any modifications that may have been made by other plug-in extension implementations. Each plug-in extension is responsible for processing all necessary input parameters, examining product configuration, and determining if there is anything for the plug-in extension to do. It is expected that many plug-in extensions will often have no work to do and will just return after making that determination.

Plug-in extension implementations are invoked along with their parent plug-in for all rehydration operations including start, stop, ping, and rehydrate. For each operation, the base plug-in will be executed first. Following successful completion of an operation executed on the plug-in, the corresponding operation will be invoked on each plug-in extension sequentially. Each plug-in extension is responsible for determining if it needs to do anything during the execution of any particular operation.

The first error encountered during a dehydration/rehydration operation in the plug-in or any plug-in extension will abort the operation and the error will be returned immediately without initiating execution of any additional plug-in extension. This means that failure of a plug-in extension will lead to failure of the entire operation.

5.5.4 Plug-in Extension Implementation

The procedures for implementing a plug-in extension are virtually identical the procedures for implementing a plug-in. The primary difference is that a plug-in extension must implement the IntrospectorPluginExtension interface instead of the IntrospectorPlugin interface.

The IntrospectorPluginExtension interface has the following 3 additional methods:

String getParentName();

Returns the name of the parent plug-in or plug-in extension that the plug-in extension extends. The name returned here must be identical to the name returned from getName() of an existing plug-in or another existing plug-in extension.

String getParentPluginName();

Returns the name of the parent plug-in. If extending directly from a plug-in, getParentName() returns the same value.

boolean isAlias();

Determines whether or not applications can present the name of this plug-in extension as a distinct choice when presenting the set of possible introspection choices. Example: when this method returns true, abctl will create a separate introspectXXX command where XXX is the name of the plug-in extension, though when users execute introspectXXX the underlying parent plug-in will be executed normally just as if the command with the name of the parent plug-in was executed.

More information about each of these methods can be found in the JavaDoc for IntrospectorPluginExtension. All the other methods on the IntrospectorPluginExtension are identical to the methods of IntrospectorPlugin.

5.5.5 Plug-In Extension SPIF Service Activation

A plug-in extension is registered as a service in the same way that a plug-in is registered with the exception that a plug-in extension must be registered using `IntrospectorPluginExtension.class.getName()` as the service name instead of `IntrospectorPlugin.class.getName()`:

```
this.registration = serviceContext.registerService(
      IntrospectorPluginExtension.class.getName(), pluginExtension);
```

See Chapter 6, "Developing an Introspection Plug-in" for more information on registering your plug-in.

5.6 Support Services

This section describes support services for Oracle Virtual Assembly Builder.

5.6.1 SPIF

The Service Provider Interface Framework (SPIF) provides an OSGi-like infrastructure that can be used directly by Java clients. This framework provides a custom class-loading mechanism, application services, general services, and service discovery. SPIF is built on top of the simple service model Java provides called SPI.

5.6.1.1 Class Loading

Within the Oracle Virtual Assembly Builder ORACLE_HOME directory structure, there is a 'jlib' directory containing the SPIF jar and a number of subdirectories. The subdirectory named 'apps' has a special meaning for the SPIF framework: it contains additional subdirectories, which define the applications provided by Oracle Virtual Assembly Builder. The other directories are there to allow us to group related modules together. This is done to make dependencies more clear, but also to allow for more fine-grained class-loading. Applications are able to specify the list of these top level jlib directories that they should load classes from. If one of these top-level jlib directories has subdirectories, these will be loaded as well. For instance, if you look at the 'packager' directory, you will see it has a sub-directory called 'plugin', and 'plugin' has additional jar files. So, adding 'packager' to an application's list of directories means that the application will load any jar file located within that directory tree.

5.6.1.2 Applications

SPIF Applications are the entry point into the Oracle Virtual Assembly Builder architecture. Plug-in developers should know that each of the Oracle Virtual Assembly Builder applications that execute plug-ins will automatically load the plug-ins that have been installed. Other jars that are required by the apps are specified via the spif.properties file.

5.6.1.3 Services

SPIF Services are a key component to developing a plug-in. A Service Activator is used to both register services as well as obtain service instances registered from other Service Activators.

A service is nothing more than an instance of a class that implements some interface. The consumers of the service don't have to know the name of the implementation class nor how to instantiate it. They know that there is an instance that has been registered as a specific interface, and they can get a reference to it and use the instance without caring how it was created.

5.6.1.4 Creating a Service

As a plug-in developer, you must create at least one service. The interface between the Introspection Framework and the plug-in is `oracle.as.assemblybuilder.introspector.plugin.IntrospectorPlugin`. This interface is located in `ORACLE_HOME/jlib/introspector/oracle.as.assemblybuilder.introspector_0.1.0.jar`. Tutorial One in Chapter 6, "Developing an Introspection Plug-in" shows an example of creating an implementation of this interface, as well as the rest of the process of connecting the implementation into the Introspection Framework.

5.6.1.5 Service Activator

A ServiceActivator implementation is used as an entry point for SPIF. It provides both a `start()` and a `stop()` method. Multiple Service Activators are allowed within a single jar file. If there are multiple Service Activators, each of them will be invoked; however, the order they are invoked is not defined. The start() method will be invoked when the jar is first loaded and before the SPIF application is started. This makes sure that all services are registered prior to application logic being invoked. The stop() method will be called after the SPIF application has stopped and before the JVM has terminated.

The Service Activator interface name is 'oracle.as.assemblybuilder.spif.ServiceActivator'. It is found in the 'AB_HOME/jlib/oracle.as.assemblybuilder.spif_0.1.0.jar' file. In order for an implementation of this class to be invoked, it first needs to be connected to SPIF. This is done by adding a file named 'META-INF/services/oracle.as.assemblybuilder.spif.ServiceActivator' to your jar. In the ServiceActivator file, add the fully-qualified class name of the Service Activator implementations. Each class name must be on a line by itself, with a new-line at the end of each line, including the last line.

When Oracle Virtual Assembly Builder loads the jar file, the activator will be loaded and the start() method will be called. The Service Activator can then be used to either register a service or locate an existing service.

5.6.1.6 Registering a Service

After creating an implementation of an interface and associating it with a Service Activator, you'll want to register it as a service.

In the start method of the Service Activator, create an instance of the class that provides the service. Then using the ServiceContext, call registerService(), passing in the name of the interface that is the service and the instance. Make sure to save the ServiceReference that is returned from this method so that in the stop() method you can call unregister() on the ServiceReference.

Here is an example of registering a service and unregistering it:

```
public class PluginActivator implements ServiceActivator {
  private ServiceRegistration registration = null;

  @Override
  public void start(ServiceContext serviceContext) throws Exception {
    IntrospectorPlugin plug-in = new SamplePlugin();

    registration =
        serviceContext.registerService(
            IntrospectorPlugin.class.getCanonicalName(),
            plugin);
  }

  @Override
  public void stop(ServiceContext serviceContext) throws Exception {
    if (this.registration != null) {
      this.registration.unregister();
    }
  }
}
```

5.6.1.7 Finding a Service

SPIF service instances may be obtained via multiple methods and interfaces provided from the ServiceContext. A recommended approach is to obtain a Collector from the ServiceContext and use the Collector to keep track of the SPIF services you need.

It is important that ServiceActivator implementations do not attempt to acquire any services from within the start() method. When this is not done, there exists a chance that a desired service will not be found. Ideally each service instance is obtained only at the point where it is needed and long after the start() method has returned.

To use a Collector, first create the Collector during execution of the start() method of your ServiceActivator:

```
myCollector = serviceContext.createCollector();
```

This Collector instance would then be passed to your own service implementation either directly or by wrapping the instance with an Interface defined in your plug-in implementation in order to hide the details of how services are obtained.

Near the point where a service (or services) are needed, one of the following Collector methods may be invoked:

- `SomeService myInstance = myCollector.getSingletonService(SomeService.class);`
- `List<SomeService> myInstances = myCollector.getServices(SomeService.class);`

Use getSingletonService() when you require exactly one instance of a given service. If more than one instance of the service has been registered than an instance will be selected arbitrarily and returned. If no instances exist than an error will be thrown.

Use getServices() when you need to acquire multiple instances of a given service. If no instances exist then the returned list will be empty.

5.6.2 Validation Framework

Plug-ins can provide validation code for various Oracle Virtual Assembly Builder lifecycle operations. This code can be shipped in the same jar as the other plug-in code and can be registered in the same ServiceActivator as the IntrospectorPlugin.

5.6.2.1 File Set Capture Validation

During the process of file set capture (previously known as "Packaging" and still referred to that way in the SDK), the files described by the file set definitions are zipped into file sets (previously "packages"). This typically happens immediately after introspection is complete, but that initial creation of file sets may be deferred, and the OVAB user may choose to recreate the file sets at a later time.

The Oracle Virtual Assembly Builder framework allows the SDK user to provide a plug-in to be invoked at the start of the file set capture process, the goal of which is to ensure that the product is in a state conducive to being captured. For instance, it may be that a product needs to be shut down before file set capture can proceed; if the validation plug-in finds that the product is running, the file set capture can be aborted. The plug-in can provide an error message to be displayed to the OVAB user.

See complete details starting at the PackagerValidationPlugin interface in the SDK.

Registration of the PackagerValidationPlugin is just the same as the registration of the introspector plug-in. The implementation of the PackagerValidationPlugin interface can even be in the same jar as the introspector plug-in, or it could be delivered in a different jar placed in the same directory.

5.6.2.2 Platform and OS Validation

A plug-in may have restrictions on the types of platform and OS it can support. The Oracle Virtual Assembly Builder framework provides a facility for enforcing these requirements after an appliance has been introspected. The JVM of the validation may be different from that of the introspection, so this validation should not rely on state information beyond what is provided in assembly metadata.

To use this facility, create a class to implement the interface PlatformOsValidationPlugin. The installation of this class is similar to the installation of the introspector class. You can deliver the class in the same jar file as the introspector plug-in or in its own jar file in the plug-in directory.

The Oracle Virtual Assembly Builder framework invokes the platform and OS validation for all implementations belonging to the plug-in and its plug-in extensions. Each such implementation provides a validation response, with this response indicating compliance or non-compliance.

Non-compliance can be communicated as either a warning or a rejection, along with a plug-in-specified error message. All warnings are relayed to the user if no rejection is indicated.

5.6.2.3 Template Validation

The Oracle Virtual Assembly Builder framework provides a facility that plug-in writers can use to ensure that any OS base image used for template creation meets the needs of the plug-in's product. Plug-ins can arrange for checks of user and group accounts, software (RPM) packages, kernel configuration parameters, and system swap space.

To use this facility, create a class that implements the interface BaseImageValidationPlugin. The installation of the class is similar to the installation of the introspector class. You can deliver the class in the same jar file as the introspector plug-in or in its own jar file in the plug-in directory.

For each validation that should be performed, the plug-in creates a Validator object describing the validation. The Oracle Virtual Assembly Builder framework traverses the list of validators during the templating process and checks that the proposed base image meets the requirements, emitting the plug-in-specified warning or error message if not.

5.6.3 Plug-in Discovery and Installation

Starting with the 12.1.2.0.0 version of Oracle Virtual Assembly Builder, the introspection plug-ins are not shipped with Oracle Virtual Assembly Builder itself, but are instead shipped and installed with the product that they support. The plug-ins are 'discovered' by Oracle Virtual Assembly Builder and installed into the Oracle Virtual Assembly Builder instance for use. In order to facilitate this, the plug-in writer has a few responsibilities:

- Install your plug-in with your product into a specific location.
- Create a plugin.config file and ship/install it along side your plug-in.

- Create an IntrospectorPluginLocator service and register it from your ServiceActivator (more info on your plug-in's ServiceActivator can be found in Chapter 6, "Developing an Introspection Plug-in").

5.6.3.1 Plug-in Location

Install your introspection plug-in or extension to the following location:

```
$ORACLE_HOME/plugins/ovab/introspector/<plugin-name>/*
```

5.6.3.2 plugin.config File

Each plug-in or extension must ship a properties file for use during plug-in discovery, initial installation, and upgrade. This file will reside in each plug-in's directory at the following location:

$ORACLE_HOME/plugins/ovab/introspector/<plug-in name>/plugin.config

The following properties should be published in the plugin.config file:

plugin.name

Name of the plugin/extension, as returned by getName() and same as the directory name. Should be indicative of the product which this plugin/extension is applied to.

plugin.version

The version of the plug-in that changes as plug-in implementation changes. This is different than the version of the product with which the plug-in ships as this version may change more or less frequently. The value must conform to the version format described below.

product.version.min

The minimum version of the product that this plug-in supports. This property is used only for informational purposes. The value must conform to the version format described below.

product.version.max

The maximum version of the product that this plug-in supports. This will usually be equivalent to the version of the product in the installation containing the plug-in. This property is used only for informational purposes. The value must conform to the version format described below.

plugin.description

A brief description of the plug-in/extension. This should include the product(s) which the plug-in/extension operates on.

ovabsdk.version

The minimum version of Oracle Virtual Assembly Builder's Introspector plug-in SDK required by this plugin/extension. This is different than the "OVAB version" and changes less frequently: Oracle Virtual Assembly Builder itself can rev without the SDK actually changing. For the 12.1.2.0.0 version of Oracle Virtual Assembly Builder, the SDK version is 3.0 - so use 3.0 in your plugin.config file.   Plug-in writers will be informed whenever the SDK version changes.

parent.plugin.name

The name of the parent plug-in or extension. Set only for extensions (top level plug-ins do not have a parent).

parent.plugin.version

The minimum version of the parent plugin/extension required by this extension. Set only for extensions (top level plug-ins do not have a parent). The value must conform to the version format described below.

Interaction with the property file will include the following behaviors:

- Oracle Virtual Assembly Builder will report an error if a property file does not exist in a `<plugin-name>` directory
- Oracle Virtual Assembly Builder will not report an error when encountering a property that is unknown. This may occur if we are looking at a plug-in home that is newer than the Oracle Virtual Assembly Builder product in use.
- Oracle Virtual Assembly Builder should document defaults when a property is not specified. This may occur if we are looking at a plug-in home that is older than the Oracle Virtual Assembly Builder product in use.

Version Number Format

All version numbers in the property file must be take the form of a simple "number-dot-number" style version. These version numbers use a syntax that consists of positive decimal integers separated by periods ".", for example "5", "2.0" or "1.2.3.4.5.6.7". This allows an extensible number to be used to represent major, minor, micro, etc. versions. The version specification is described by the following formal grammar:

```
DotVersion:
    Digits RefinedVersion
RefinedVersion:
    . Digits
    . Digits RefinedVersion
Digits:
    Digit
    Digits
Digit:
    any character for which Character.isDigit(char) returns true, for example, 0, 1, 2, ...
```

Version numbers are compared by sequentially comparing corresponding components of the version number. Each component is treated as an integer and the values compared. If the value is greater than the desired value true is returned. If the value is less false is returned. If the values are equal the next pair of components is compared. Where one version has more components than the other the missing components are assumed to be 0 and treated as if they were present during comparison.

5.6.4 Implementing an IntrospectorPluginLocator Service

Plug-ins and extensions are, by definition, capable of determining the product root directory of their own product from the job parameters supplied supplied by the user at introspection. Often the product root directory is an explicit job parameter though other times it is derivable from one or more of the specified job parameters. The Introspector Framework is not capable of knowing how to derive this information from the job parameters it passes to plug-ins and extensions, yet it needs to know this information in order to be able to automatically check for plug-in and extension updates at the beginning of each introspection.

To resolve this issue plug-ins and extensions should publish an IntrospectorPluginLocator service implementation.

At the beginning of every introspection (dehydration) the IntrospectorFramework takes the following steps:

1. Determine the plug-in and any extensions participating in the introspection.
2. Look for IntrospectorPluginLocator implementations that return the same names from their `getPluginOrExtensionName()` method.
3. Build a list of product roots to search by executing the `getProductInstallHome()` method on each IntrospectorPluginLocator implementation found in step 2. Duplicates are consolidated.
4. Search in the product roots derived in step 3 for newer versions of the plug-in and extensions found in step 1. If any are found then abort the introspection and report the discovered upgrade candidates back to the user.

IntrospectorPluginLocator implementations are tracked as SPIF services and may be registered in the same manner and from within the same ServiceActivator implementation used to register the corresponding plug-in or extension.

5.6.5 Plug-in Backward Compatibility Requirements

When installing a plug-in or extension, the information in plugin.config is consulted to determine if the plug-in or extension is compatible with what is already installed. It is assumed that it is always safe to update an older version with a newer version. In order for this to work, both plugins and extensions are expected to be backward compatible according to the following definition (applies to both plugins and extensions):

- A plug-in is assumed to always be backward compatible with respect to the product versions it can dehydrate/rehydrate. In other words, a newer plug-in shipped with a new product must continue to dehydrate/rehydrate the older versions of products it was previously capable of capturing.
- A plug-in is assumed to always be backward compatible with respect to the catalog metadata it can process. In other words, a newer plug-in must continue to be able to rehydrate a product using metadata created by prior versions of itself.
- A plug-in is assumed to always be backward compatible with respect to how it interacts with extensions that extend from it. An extension developed against one version of a parent must still continue to work against newer versions of the same parent.

5.6.6 Plug-in Installation Guidelines

Operations related to plug-in and extension installations must adhere to the following guidelines:

- An extension may not be installed without the parent already installed.

- A plug-in may not be downgraded to a version older than the version installed unless no appliance created by the plug-in exists. Doing so may break appliances created by the plugin.
- A plug-in may not be deleted when an appliance created by the plug-in exists. Doing so would break appliances created by the plugin.
- The deletion of a plug-in also deletes recursively all children of the plugin.
- Disabling a plug-in also disables recursively all children of the plugin.
- An extension may not be enabled if its parent is disabled.

These guidelines are enforced by the various plug-in installation operations provided by Oracle Virtual Assembly Builder applications.

5.7 Utilities and Helpers

The plug-in SDK provides some utilities and helper classes for common tasks that may need to be performed by more than one plug-in. The introspector.plug.util package contains UtilFactory, which is registered as a SPIF service. You can get a handle to this service by using the following in your plug-in's SPIF service activator class:

```
UtilFactory utilFactory =
    (UtilFactory) myCollector.getSingletonService(UtilFactory.class.getName());
```

5.7.1 OCM Registration

`UtilFactory.getOcmUtil()`

Use this utility to add Oracle Configuration Manager related properties to an appliance during dehydration, and to register the appliance with OCM during rehydration based on those properties.

5.7.2 OUI Central Inventory

`UtilFactory.getOuiUtil()`

Use this utility to create the metadata necessary for constructing an OUI Central Inventory and registering the ORACLE_HOME(s) in that inventory during deployment.

5.7.3 SSH

`UtilFactory.openSshConnection()`

Use the SshConnection class if you need to SSH to another machine to run a command or send/retrieve files.

5.7.4 Velocity

`UtilFactory.getContentBuilder()`

Velocity is a utility for doing token replacement in files. The ContentBuilder class helps you make a content resource with optional token replacement performed by Velocity.

The `RehydrateUtil` class also has some Velocity related methods that can be used during rehydration to fix up config files.

5.7.5 Misc Utilities

`UtilFactory.getRehydrateUtil()`

You can find various utilities for copying files and running commands in the `RehydrateUtil` class.

5.8 Best Practices

Here are some best practices, caveats and things to keep in mind while designing and developing your introspector plug-in.

5.8.1 Avoid Modifying the Reference System

Your plug-in should not modify the reference host during introspection. Temporary files should not be created under the reference product's directories. The reference product configuration and runtime state should not be altered.

If you need to create temporary files, use the Oracle Virtual Assembly Builder `tmp` directory. The `AbConfig` class has a method for finding the location of the Oracle Virtual Assembly Builder `tmp` directory.

5.8.2 Avoid Static Variables

In Oracle Virtual Assembly Builder Studio your introspector plug-in gets instantiated once and is potentially used repeatedly to do introspection and file set capture operations. You should not use static variables as this may cause unexpected results if multiple introspections of your product are performed in one Studio session.

5.8.3 Do not Attempt to Access SPIF Services during ServiceActivator.start()

While calling `ServiceActivator.start()`, do not attempt to access SPIF services.

5.8.4 Reserved Names and Naming Restrictions

There are certain restrictions on the names your plug-in can give to the different metadata elements.

5.8.4.1 Appliance and Assembly Names

Appliance and Assembly names must contain only ASCII letters (A-Z a-z), digits (0-9), and these symbols: ! # % ( ) * + , - . : = ? @ [ ] ^ _ { } ~

Name length must be 40 characters or less.

5.8.4.2 Property Names

Property names have a similar character set restriction to Appliance and Assembly names: ASCII letters (A-Z a-z), digits (0-9), and these symbols: ! # % ( ) * + , - . : = ? @ [ ] ^ _ { } ~

There is currently no length restriction for property names, but one may be added in the future.

Do not create any property names that start with 'OVAB', 'Ovab' or 'ovab' as those names are reserved for internal use.

5.8.4.3 Type

The appliance type should not be set to anything starting with 'External', as types that start with that string are reserved for internal use.

5.8.5 Managing External Dependencies

Plug-in developers should be aware that all Oracle Virtual Assembly Builder services, including introspector plug-ins, are loaded by a single classloader. This can create conflicts when the same class is supplied by multiple plug-ins; the plug-ins may not expect to share the class. Plug-ins that supply external jars, that also might be included by other plug-ins, should use the ImplProvider facility described below to load those jars.

A plug-in may need to make use of jars from the target product, for instance to examine or change product configuration using a proprietary API shipped with the product. Using product jars within a plug-in introduces additional maintenance issues because the plug-in code using these APIs may need to change to accommodate new product versions and changed APIs.

Often the use of product jars can be avoided with a little effort but if you determine that you must make use of jars from the target product then there are a couple of options available to you.

The first option is to add those jars to the directory where the plug-in is deployed. The downside of this approach is that the external product jars need to ship both with the product and with Oracle Virtual Assembly Builder. Also, as the product version changes multiple copies of the product jars may need to ship with Oracle Virtual Assembly Builder in order for the plug-in to support each new product version.

The second option is to load the product jars dynamically at runtime from where they are installed with the product. This approach requires a little more effort but eliminates the need to ship product jars (potentially multiple versions!) with Oracle Virtual Assembly Builder. This is the recommended approach.

Ideally, an introspection plug-in or plug-in extension would be able to interact with multiple versions of a product. This becomes more of a challenge in the case where product jars need to be used, as the APIs in those product jars may change from one version to the next. There are multiple solutions to this but we have provided one potential path worth considering.

The basic idea is to load the necessary product jars from the installation dynamically during execution and also create an abstraction layer through which those jars are accessed. Here is an outline of what can be done to use this approach:

1. Create a "wrapper" interface that abstracts the tasks performed using the product jars. This abstraction should be at a high enough level to be applicable across product versions. The interface(s) can reside in the plug-in's jar that always gets loaded.
2. Create an implementation of the above interface for each product version where the product API has changed. Initially only one such implementation would be created and more would be added as the product's API changes in subsequent product releases. These implementations can either reside in a jar file within the product installation or can reside in a separate jar within the Oracle Virtual Assembly Builder installation. In either case, the implementation should get loaded dynamically at run time.
3. At run time, determine the product version and load the corresponding "wrapper" implementation plus the product jars needed by that implementation.

To facilitate this approach:

- Oracle Virtual Assembly Builder, at startup, skips loading any jars located in directories named "dynlib" found anywhere within Oracle Virtual Assembly Builder's ORACLE_HOME/jlib/ directory. Multiple "wrapper" implementations can be put into this directory and can be loaded dynamically as needed.
- UtilFactory.createImplProvider() has been provided to assist in dynamically loading "wrapper" implementations and the necessary jars from product installations. Take a look at the javadoc for this method and ImplProvider in the oracle.as.assemblybuilder.introspector.plugin.util package.

Hypothetical example:

```
/**
 * Interface for interactions with MDS.
 *
 * Resides in jlib/stuff/myplugin.jar
 */
interface MdsWrapper {
  String getMdsFoo();
  String getMdsBar();
  void setMdsFoo(String value);
  void setMdsBar(String value);
}

/**
 * Uses 11g APIs to interact with MDS (fabric-core.jar).
 *
 * Resides in jlib/stuff/dynlib/mdswrapper-v1.jar
 */
class MdsWrapperImplVersion1 implements MdsWrapper {
  ...
}

/**
 * Uses 12g APIs to interact with MDS.
 *
 * Resides in jlib/stuff/dynlib/mdswrapper-v2.jar
 */
class MdsWrapperImplVersion2 implements MdsWrapper {
  ...
}
```

Given the above code the plug-in could obtain the needed implementation according to the product version:

```
  void doMdsStuff(String oracleHome) {

    String fabricJarPath = oracleHome + "/jlib/fabric-core.jar";

    String productVersion = getVersion(oracleHome);

    MdsWrapper mdsWrapper = null;

    // obtain product version specific implementation
    if (productVersion.equals(VER_11G)) {
      ImplProvider implProvider =
          utilFactory.createImplProvider("mdswrapper-v1.jar", fabricJarPath);
      mdsWrapper =
          implProvider.createInstance(MdsWrapper.class, "MdsWrapperImplVersion1");
    }
    else if (productVersion.equals(VER_12G)) {
      ImplProvider implProvider =
          utilFactory.createImplProvider("mdswrapper-v2.jar", fabricJarPath);
      mdsWrapper =
          implProvider.createInstance(MdsWrapper.class, "MdsWrapperImplVersion2");
    }

    // use implementation generically
    String foo = mdsWrapper.getMdsFoo();
    String bar = mdsWrapper.getMdsBar();
    mdsWrapper.setMdsFoo(foo);
    mdsWrapper.setMdsBar(bar);
  }
```

5.9 Testing your Introspector Plug-in

Describes testing of the dehydration and rehydration functions of your plug-in.

5.9.1 Dehydration Testing

Testing the dehydration functionality in your introspector plug-in is straightforward - simply perform an introspection. Refer to the Section 5.6.3, "Plug-in Discovery and Installation" for information on how to get your plug-in installed into the Oracle Virtual Assembly Builder environment.

5.9.2 Rehydration Testing and Troubleshooting

Testing the rehydration functionality in an introspection plug-in can be time consuming and trouble shooting in the OVM environment can be a challenge. This section provides guidance. See also "Troubleshooting" in *Using Oracle Virtual Assembly Builder*.

5.9.2.1 Deployment Lifecycle

Understanding the steps that occur when Oracle Virtual Assembly Builder deploys an assembly can help you triage failures. There are several actions taken on the VM before your plug-in code is even invoked.

1. Virtual Machine Creation and Start

   The Oracle Virtual Assembly Builder Deployer instructs OVM manager to create virtual machines for all appliances in the assembly you are deploying. After all the virtual machines are created they are started based on the dependencies between the appliances.

2. AB Service Creation on the VM

   When the VM starts up, a special script called oraclevm-template.sh is automatically run. This script installs a Linux service called 'ab'. The ab service is responsible for network configuration and for launching the 'Appliance Deploy Driver'.

3. VM Network Configuration

   The ab service is started by the oraclevm-template.sh script and configures the network settings on the VM.

4. Late Bindings Retrieved through Phone Home

After the network setup is complete the ab service launches a Java process: the 'Appliance Deploy Driver'. This Java code handles communication to the Oracle Virtual Assembly Builder Deployer to retrieve the late bindings. It also launches the rehydrator code which calls the plug-ins, and then contacts the Oracle Virtual Assembly Builder Deployer again to report the success or failure of the rehydration.

The Appliance Deploy Driver contacts the Oracle Virtual Assembly Builder Deployer to retrieve the late binding information.

Once the late binding information has been retrieved from the Oracle Virtual Assembly Builder Deployer, the metadata on the VM is updated to reflect the late bindings. This updated metadata is under `/assemblybuilder/catalog/metadata/root`.

5. Introspector Plug-in Invoked

   After the late bindings are saved to the catalog on the VM (which is a subset of the catalog on your Oracle Virtual Assembly Builder host, containing only as much metadata as is needed to rehydrate the current appliance being deployed), the plug-in's `Rehydrator.rehydrate()` method is invoked.

6. Oracle Virtual Assembly Builder Deployer is sent rehydration result

   If the base plug-in and all its extensions complete successfully, the Appliance Deploy Driver sends the successful result to the Oracle Virtual Assembly Builder Deployer. If there is a failure at any point (the base plug-in or any extension throws an exception) then the failure result is sent when the failure occurs.

6 Developing an Introspection Plug-in

This chapter describes the usage of the Oracle Virtual Assembly Builder introspection API through a set of code tutorials for the development of an Oracle Virtual Assembly Builder introspector plug-in. This chapter contains the following sections:

6.1 Introduction

The goal of this document is to provide a step-by-step tutorial for building a plug-in.

The tutorial source for the examples in this cookbook are available in the SDK in a 'tutorials' directory.

This document is not be written for any particular development environment, and assumes you understand how to perform the given steps in your development environment.

6.2 Tutorial 1: Oracle Virtual Assembly Builder Integration

The purpose of this tutorial is to create the most simple plug-in. The goal of this tutorial is to show you how to register a plug-in with the Oracle Virtual Assembly Builder architecture. This tutorial is the basis of all subsequent tutorials.

This tutorial covers:

- The IntrospectorPlugin interface
- The ServiceActivator interface
- Registering the IntrospectorPlugin
- Building the tutorials
- Installing the IntrospectorPlugin

- Verifying the plug-in was deployed

6.2.1 Creating a Project

To start, create an empty Java project called 'SamplePlugin'. You must use JDK 1.6 for these tutorials since Oracle Virtual Assembly Builder is not compatible with earlier versions of Java.

Add the following jar files from the SDK to your project:

- jlib/oracle.as.assemblybuilder.spif_0.1.0.jar
- jlib/introspector/oracle.as.assemblybuilder.introspector.plugin.api_0.1.0.jar
- jlib/core/oracle.as.assemblybuilder.external.common_0.1.0.jar

6.2.2 Creating an IntrospectorPlugin

Create a plug-in. The interface 'oracle.as.assemblybuilder.introspector.plugin.IntrospectorPlugin' defines a plug-in. Create the class 'sample.plugin.internal.SamplePlugin' that implements the Introspector Plugin interface. In this case 'internal' implies that any classes here are not for others to use.

You should now have a class that has the following methods: `getName()`, `getJobParameters()`, `initialize()`, `getDehydrator()`, and `getRehydrator()`. For now, all you need to do is change the return value of `getName()` to be 'SamplePlugin' and make `getJobParameters()` return an empty list. The rest of the changes to the plug-in implementation are covered in the subsequent tutorials. The class appears as follows:

```
package sample.plugin.internal;

import java.util.ArrayList;
import java.util.List;

import oracle.as.assemblybuilder.introspector.job.JobParameter;
import oracle.as.assemblybuilder.introspector.job.JobParameterBuilder;
import oracle.as.assemblybuilder.introspector.plugin.IntrospectorPlugin;
import oracle.as.assemblybuilder.introspector.plugin.dehydrate.Dehydrator;
import oracle.as.assemblybuilder.introspector.plugin.rehydrate.Rehydrator;

public class SamplePlugin implements IntrospectorPlugin {

  @Override
  public Dehydrator createDehydrator() {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public Rehydrator createRehydrator() {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public List<JobParameter> getJobParameters() {
    List<JobParameter> jobParameters = new ArrayList<JobParameter>();

    return jobParameters;
  }

  @Override
  public String getName() {
    return "SamplePlugin_Tutorial1";
  }

  @Override
  public void initialize(JobParameterBuilder jobParameterBuilder){
    // TODO Auto-generated method stub
  }
}
```

6.2.3 SPIF Service Activation

To allow the plug-in to be seen by the Oracle Virtual Assembly Builder framework, start by creating a Service Activator. A Service Activator is part of the SPIF framework. If you have not done so already, read about SPIF in Chapter 5, "Preparing to Develop using the Oracle Virtual Assembly Builder Plug-in SDK".

Create a new class 'sample.plugin.internal.spif.PluginActivator' that implements 'oracle.as.assemblybuilder.spif.ServiceActivator'.

To get the service activator to be called, you need to register it. Create a folder in the root project directory called 'resources/jar/META-INF/services'. In that folder, create a text file called 'oracle.as.assemblybuilder.spif.ServiceActivator'. In that file put a single line of text that is the name of the Service Activator implementation, making sure to put a new-line at the end of the line. In this case the line should be 'sample.plugin.internal.spif.PluginActivator'. You also need to make sure that this file gets added to your built jar file in the 'META-INF/services' directory. Once this is done, the `start` method on the Service Activator is invoked when one of the Oracle Virtual Assembly Builder user interfaces start-up and the `stop` method is invoked when they are shut-down.

At this point we have a Service Activator that gets called, but the Service Activator does not yet register the sample plug-in as an Introspector plug-in. To complete the activation, change the `PluginActivator` as follows:

```
package sample.plugin.internal.spif;

import oracle.as.assemblybuilder.introspector.plugin.IntrospectorPlugin;
import oracle.as.assemblybuilder.spif.ServiceActivator;
import oracle.as.assemblybuilder.spif.ServiceContext;
import oracle.as.assemblybuilder.spif.ServiceRegistration;
import sample.plugin.internal.SamplePlugin;

public class PluginActivator implements ServiceActivator {
  private ServiceRegistration registration = null;

  @Override
  public void start(ServiceContext serviceContext) throws Exception {
    IntrospectorPlugin plugin = new SamplePlugin();

    this.registration = serviceContext.registerService(
      IntrospectorPlugin.class.getCanonicalName(),
      plugin);
  }

  @Override
  public void stop(ServiceContext serviceContext) throws Exception {
    if (this.registration != null) {
      this.registration.unregister();
    }
  }
}
```

In the `start` method, you create an instance of the plug-in, then we register it as an `IntrospectorPlugin`. After the `registerService` call, the Introspector Manager is notified that there is a new plug-in and you can then invoke the plug-in from the Oracle Virtual Assembly Builder user interfaces. Of course, at this point we have not implemented job parameters, the dehydrator, nor the rehydrator so nothing interesting will occur.

In the `stop` method, we make sure to unregister the service we registered.

6.2.4 Building the Tutorials

Before you can install and use your plug-in you need to create a jar file for your plug-in classes. Most development environments provide a way to export a project to a JAR file. Make sure the META-INF/services/oracle.as.assemblybuilder.spif.ServiceActivator file is added to this JAR along with the Java classes.

You can also build the tutorials and sample plug-ins provided with the Oracle Virtual Assembly Builder SDK using ant. Navigate to the tutorial or sample directory and enter 'ant dist'. This command builds a jar containing the classes and the appropriate ServiceActivator file and places it in the necessary directory structure for discovery and installation.

6.2.5 Installing the Plug-in

Once you have a jar file, you need to create a mock ORACLE_HOME directory structure along with a plugin.config file.

In Oracle Virtual Assembly Builder, plug-ins must be installed using the 'installPlugins' command. For this command to work, you need to create a mock ORACLE_HOME, which we'll call 'ab_home', that has a 'plugins/ovab/introspector/<plugin-name>' directory, where <plugin-name> is replaced by the value you return from the 'getName()' method of the plug-in. In this case, the directory would be 'ab_home/plugins/ovab/introspector/SamplePlugin_Tutorial1'.

Inside that directory, you need to place your plug-in JAR file along with a 'plugin.config' file. The 'plugin.config' file requires the following contents:

```
plugin.name=SamplePlugin
plugin.version=1.0.0
product.version.min=1.0.0
product.version.max=2.0.0
plugin.description=My sample plugin.
ovabsdk.version=3.0
```

With the jar and plugin.config file in place, you're now ready to install the plug-in into an Oracle Virtual Assembly Builder installation. This is done using the 'installPlugins' `abctl` command. For example:

```
./abctl installPlugins -productRoot /path/to/ab_home
```

6.2.6 Executing an Introspector Plug-In

Describes executing an introspector plug-in using `abctl` or Studio.

6.2.6.1 abctl Command-Line User Interface

Once you install a plug-in, the CLI automatically registers a command with the name introspect*`plug-in name`*. For example, if you have a Plug-In called 'SamplePlugin', the command you see when you run 'abctl help' shows 'introspectSamplePlugin'.

Once installed, then you can run the introspection for the plug-in.

You can change into the AB_INSTANCE/bin directory and run './abctl help introspection'. The result is output similar to the following:

```
> ./abctl help introspection

Usage: abctl command [options]

Command                 Description
----------------------------------------------------------------------------
captureFileSets         Creates file sets for specified appliance or assembly.
describePlugins         Lists the installed plugins.
disablePlugin           Disables a plugin or extension.
enablePlugin            Enables a plugin or extension.
findPlugins             Finds plugins to install.
installPlugins          Installs 1 or more plugins.
introspectCoherenceWeb  Alias for command "introspectWLS".
introspectGenericProd   Examines GenericProd configuration and captures
                        metadata.
introspectOHS           Examines OHS configuration and captures metadata.
introspectSamplePlugin  Examines SamplePlugin configuration and captures
                        metadata
introspectWLS           Examines WLS configuration and captures metadata.
removePlugin            Removes a plugin or extension.

Try "abctl help name" for detailed help of a specific command.
```

The plug-in gets its own introspect command automatically, just by registering it as a plug-in. You can see it above in bold. If you were to type './abctl help introspectSamplePlugin' you would get help about all of the parameters this plug-in supports. Even though we have not defined any JobParameters, a number of parameters will be listed. The CLI command that launches the plug-in has its own parameters and all plug-ins will have these as well. Any JobParameters you add to the plug-in are added to this list. You can also type './abctl introspectSamplePlugin' to initiate a dehydration job. However, since `getDehydrator()` returns null, you get the following error:

```
Launching introspection of component 'SamplePlugin' ...
Task is done: DehydrateJob failed with error: null
Error: OAB-7105: Introspection failed
```

6.2.6.2 Oracle Virtual Assembly Builder Studio

Installed plug-ins automatically appear within the graphical user interface as an additional plug-in. You can execute it as you would any other plug-in. For more information, see *Using Oracle Virtual Assembly Builder*.

6.2.7 Conclusion

That concludes this tutorial. You created an empty implementation of a plug-in, registered the plug-in with Oracle Virtual Assembly Builder, installed the plug-in, and confirmed it is visible from the `abctl` CLI user interface.

6.3 Tutorial 2: Dehydration

Now that you have a plug-in created and installed, you need to make it do something useful. Start by looking at *dehydration*. Dehydration is the process of collecting the metadata for a component so that it can be rehydrated later.

Start with the project from Tutorial 1. Before you can start creating the Dehydrator implementation, you need to add the Progress module to the project. Add the oracle.as.assemblybuilder.progress.api_0.1.0.jar file from the SDK jlib/core directory to the project's build path.

Next, create an implementation of the oracle.as.assemblybuilder.introspector.Dehydrator interface called 'sample.plugin.internal.DehydratorImpl'. This interface only has two methods: `initialize()` and `dehydrate()`. The `dehydrate()` method is called when a Dehydration Job is started through either the introspect<name> command of the CLI, or the from within the GUI. The private `initializeAppliance()` method is used to set the minimum amount of metadata required for a successful Dehydrate Job to be executed.

The DehydratorImpl class should appear as follows:

```
package sample.plugin.internal;

import java.util.ArrayList;
import java.util.Collection;

import oracle.as.assemblybuilder.introspector.metadata.appliance.Appliance;
import oracle.as.assemblybuilder.introspector.plugin.dehydrate.DehydrateContext;
import oracle.as.assemblybuilder.introspector.plugin.dehydrate.Dehydrator;
import oracle.as.assemblybuilder.progress.ProgressManagerFactory;

public class DehydratorImpl implements Dehydrator {

  DehydratorImpl() {
  }

  @Override
  public void initialize(ProgressManagerFactory progressManagerFactory) {
    // TODO Auto-generated method stub
  }

  @Override
  public void dehydrate(DehydrateContext context) throws Exception {
    Appliance appliance = context.getOrCreateRootAppliance();

      // initialize the appliance
    initializeAppliance(appliance);

    //TODO: This is where you will make the changes to the Appliance
  }

  /**
   * Set the minimum amount needed to be a valid appliance.
   *
   * @param appliance the appliance to update
   */
  private void initializeAppliance(Appliance appliance) {
    appliance.setVersion(1, 0, 0);
    appliance.setScalabilityInfo(1, 1, 99, 200);
    builder.addResourceRequirement(ResourceEnum.MEMORY_MB, 256);
    builder.addResourceRequirement(ResourceEnum.NUMBER_CPUS, 1);
  }
}
```

Update the plug-in to return a new instance of this class in the `createDehydrator()` method. Make sure to return a new instance as this greatly simplifies handling concurrent Dehydrate Jobs. Also, avoid using static variables to hold anything that is instance specific. Doing so may lead to problems with concurrent execution that are difficult to diagnose.

Your plug-in class should now appear as follows:

```
package sample.plugin.internal;

import java.util.ArrayList;
import java.util.List;

import oracle.as.assemblybuilder.introspector.job.JobParameter;
import oracle.as.assemblybuilder.introspector.job.JobParameterBuilder;
import oracle.as.assemblybuilder.introspector.plugin.IntrospectorPlugin;
import oracle.as.assemblybuilder.introspector.plugin.dehydrate.Dehydrator;
import oracle.as.assemblybuilder.introspector.plugin.rehydrate.Rehydrator;

public class SamplePlugin implements IntrospectorPlugin {

  @Override
  public Dehydrator createDehydrator() {
    return new DehydratorImpl();
  }

  @Override
  public Rehydrator createRehydrator() {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public List<JobParameter> getJobParameters() {
    List<JobParameter> jobParameters = new ArrayList<JobParameter>();

    return jobParameters;
  }

  @Override
  public String getName() {
    return "SamplePlugin_Tutorial2";
  }

  @Override
  public void initialize(JobParameterBuilder jobParameterBuilder) {
  }
}
```

This example uses an appliance rather than an assembly, but to create an assembly rather than an appliance change the single line that calls `getOrCreateRootAppliance()` to `getOrCreateRootAssembly()`.

Once you build your plug-in JAR, you then need to install the plug-in as described in Section 6.2, "Tutorial 1: Oracle Virtual Assembly Builder Integration".

6.4 Tutorial 3: Rehydration

Adding a Rehydrator implementation is a matter of implementing the Rehydrator interface and having the plug-in create a new instance of it when `createRehydrator()` is called.

Create a class called RehydratorImpl in the `sample.plugin.internal` package:

```
package sample.plugin.internal;

import oracle.as.assemblybuilder.introspector.plugin.rehydrate.RehydrateContext;
import oracle.as.assemblybuilder.introspector.plugin.rehydrate.Rehydrator;
import oracle.as.assemblybuilder.progress.ProgressManagerFactory;

public class RehydratorImpl implements Rehydrator {

  @Override
  public void initialize(ProgressManagerFactory progressManagerFactory) {
    // TODO Auto-generated method stub
  }

  @Override
  public boolean ping(RehydrateContext rehydrateContext) throws Exception {
    // TODO Auto-generated method stub
    return false;
  }

  @Override
  public void rehydrate(RehydrateContext rehydrateContext) throws Exception {
    // TODO Auto-generated method stub
  }

  @Override
  public void start(RehydrateContext rehydrateContext) throws Exception {
    // TODO Auto-generated method stub
  }

  @Override
  public void stop(RehydrateContext rehydrateContext) throws Exception {
    // TODO Auto-generated method stub
  }
}
```

The RehydrateContext provides access to the metadata as you created it in the Dehydrator implementation, with any changes the user made to the data either in the Oracle Virtual Assembly Builder editor or through a deployment plan. Then you must add the implementation to mutate the configuration as needed.

The RehydrateContext also provides a method for determining what sort of stop is taking place when the Rehydrator `stop()` method is called. Plug-ins can take different actions depending on whether this is a regular stop, a stop as part of a scale down operation (implying the component is involved in clustering of some kind), or a stop as part of an undeploy operation.

Any files you added in the Package Definition have already been put in place prior to the `rehydrate()` method being called.

6.5 Tutorial 4: Progress

During Dehydration, it is important to implement Progress so that the user who is waiting for Dehydration to complete will know that it is still executing. There are two types of Progress: structured and ad-hoc.

Structured progress is used to describe steps that you know will always occur during processing. The value of structured progress is that it allows the user to see all of the steps that will occur during execution of the Dehydrate Job and which step is currently executing. The downside is that you can not include specific information in your messages relating to the instance being Dehydrated. Structured progress has a tree-like structure. A top-level progress event can have children. The nesting can be as many levels deep as needed.

Ad-hoc progress is progress that is used in cases where the message content cannot be known until runtime. Ad-hoc progress is always related to a Structured Progress Event. Ad-hoc progress cannot have children.

Let's make up an example so we can make some sense out of the two different types. Let's assume that the sample plug-in has four different top-level steps: OUI processing, OPMN processing, config parsing, and config processing. OUI processing and OPMN processing are distinct events, but config parsing and config processing have sub-events. Let's assume that config processing consists of processing an unknown number of files, and we want to send a progress message for every file found. Doing this will require Ad-hoc progress. However, the config processing consists of four distinct processors. So, for config processing we will be able to utilize Structured Progress.

For this example, we need to build two separate ProgressResourceBundles. One for the top level list, we'll call this TopLevelProgressBundle. One for the config processing list, we'll call this ConfigProcessingProgressBundle. For these Structured Progress lists, we must extend the ProgressResourceBundle class, as they maintain order and order is important. The order in which you list the events in the bundles is the order in which the events are expected to be sent. Any events that get skipped will be automatically set to 'done' when you send a later event.

To handle the Ad-hoc events within the config parser, we must extend a standard ListResourceBundle.

```
package sample.plugin.resources;

import oracle.as.assemblybuilder.progress.ProgressResourceBundle;

public static class TopLevelProgressBundle extends ProgressResourceBundle {
  private static final long serialVersionUID = -2753027497921148834L;

  static public final String PROGRESS_START = "SPI-001";
  static public final String OUI_PROCESSING = "SPI-002";
  static public final String OPMN_PROCESSING = "SPI-003";
  static public final String CONFIG_PARSING = "SPI-004";
  static public final String CONFIG_PROCESSING = "SPI-005";
  static public final String PROGRESS_COMPLETE = "SPI-006";

  /**
   * Contents of key/value progress messages
   */
  static final Object[][] contents =
      { { PROGRESS_START, "dehydration starting" },
        { OUI_PROCESSING, "OUI processing completed" },
        { OPMN_PROCESSING, "OPMN processing completed" },
        { CONFIG_PARSING, "config parsing completed" },
        { CONFIG_PROCESSING, "config processing completed" },
        { PROGRESS_COMPLETE, "dehydration complete" } };

  @Override
  protected Object[][] getContents() {
    return contents;
  }
}


package sample.plugin.resources;

import oracle.as.assemblybuilder.progress.ProgressResourceBundle;

public static class ConfigProcessingProgressBundle extends ProgressResourceBundle
{
  private static final long serialVersionUID = -2105678590437947371L;

  static public final String PROCESS_STEP_1 = "SPI-101";
  static public final String PROCESS_STEP_2 = "SPI-102";
  static public final String PROCESS_STEP_3 = "SPI-103";
  static public final String PROCESS_STEP_4 = "SPI-104";

  /**
   * Contents of key/value progress messages
   */
  static final Object[][] contents =
      { { PROCESS_STEP_1, "step 1 completed" },
        { PROCESS_STEP_2, "step 2 completed" },
        { PROCESS_STEP_3, "step 3 completed" },
        { PROCESS_STEP_4, "step 4 completed" } };

  @Override
  protected Object[][] getContents() {
    return contents;
  }
}

package sample.plugin.resources;

import java.util.ListResourceBundle;

public static class ConfigParsingAdHocProgressBundle extends ListResourceBundle {

  static public final String PROCESSING_FILE = "SPI-201";

  /**
   * Contents of key/value progress messages
   */
  static final Object[][] contents =
      { { PROCESSING_FILE, "processing file {0}" } };

  @Override
  protected Object[][] getContents() {
    return contents;
  }
}
```

The rest of the work will be within the DehydratorImpl class created previously. In the `initialize()` method, save a reference to the progressManagerFactory into an instance variable.

In the following example, pay particular attention to the `initializeProgress()` method. We will discuss that method after the example.

```
package sample.plugin.internal;

import java.util.ArrayList;
import java.util.Collection;

import oracle.as.assemblybuilder.introspector.metadata.appliance.Appliance;
import

oracle.as.assemblybuilder.introspector.metadata.attributes.ResourceRequirement.Res
ourceEnum;
import oracle.as.assemblybuilder.introspector.plugin.dehydrate.DehydrateContext;
import oracle.as.assemblybuilder.introspector.plugin.dehydrate.Dehydrator;
import oracle.as.assemblybuilder.progress.AdHocMarker;
import oracle.as.assemblybuilder.progress.ProgressListener;
import oracle.as.assemblybuilder.progress.ProgressManager;
import oracle.as.assemblybuilder.progress.ProgressManagerFactory;
import oracle.as.assemblybuilder.progress.ProgressProxy;
import sample.plugin.resources.Resources;

public class DehydratorImpl implements Dehydrator {
  private ProgressManagerFactory progressManagerFactory = null;
  private AdHocMarker configParsingAdHocMarker = null;

  DehydratorImpl() {
  }

  @Override
  public void initialize(ProgressManagerFactory progressManagerFactory) {
    this.progressManagerFactory = progressManagerFactory;
  }

  @Override
  public DehydrateResult dehydrate(DehydrateContext context) throws Exception {
    Appliance appliance = context.getOrCreateRootAppliance();

    //configure all the progress proxies
    ProgressProxy progress = initializeProgress(context.getProgressListener());

    progress.onProgress(Resources.TopLevelProgressBundle.PROGRESS_START);

    //initialize the appliance
    initializeAppliance(appliance);

    //TODO: Do OUI stuff
    progress.onProgress(Resources.TopLevelProgressBundle.OUI_PROCESSING);

    //TODO: Do OPMN stuff
    progress.onProgress(Resources.TopLevelProgressBundle.OPMN_PROCESSING);

    //Just for fun, create a list of files to parse
    String[] files = { "file.1", "file.2", "file.3", "file.4" };
    for (String file : files) {
      //TODO: do something with the files

      //generate ad-hoc messages
      progress.onProgress(this.configParsingAdHocMarker,
          Resources.ConfigParsingAdHocProgressBundle.PROCESSING_FILE, file);
    }

    //Now send the processing events
    progress.onProgress(Resources.ConfigProcessingProgressBundle.PROCESS_STEP_1);
    progress.onProgress(Resources.ConfigProcessingProgressBundle.PROCESS_STEP_2);
    progress.onProgress(Resources.ConfigProcessingProgressBundle.PROCESS_STEP_3);
    progress.onProgress(Resources.ConfigProcessingProgressBundle.PROCESS_STEP_4);

    //all done
    progress.onProgress(Resources.TopLevelProgressBundle.PROGRESS_COMPLETE);
  }

  private ProgressProxy initializeProgress(ProgressListener progressListener) {
    ProgressManager progressManager =
      this.progressManagerFactory.createManager(progressListener);

    //assign the top level events to this manager
    progressManager.registerBundle(new Resources.TopLevelProgressBundle());

    ProgressManager childProgressManager = null;

    //create a child progress manager for the config parsing event
    // and make it ad-hoc,  we need to save the adHocMarker when
    // sending ad-hoc progress events
    childProgressManager = progressManager.createChildManager(
                   Resources.TopLevelProgressBundle.CONFIG_PARSING);
    this.configParsingAdHocMarker = childProgressManager.registerAdHoc(
                   new Resources.ConfigParsingAdHocProgressBundle());

    //create the child-progress events for the config processing event
    ProgressManager configProcessingProgressManager =
progressManager.createChildManager(
                   Resources.TopLevelProgressBundle.CONFIG_PROCESSING);
    configProcessingProgressManager.registerBundle(
                   new Resources.ConfigProcessingProgressBundle());

    //at this point, the progress tree is configured
    //now we generate the proxy that is used to send events

    return progressManager.generateProxy();
  }

  /**
   * Set the minimum amount needed to be a valid appliance.
   *
   * @param appliance the appliance to update
   */
  private void initializeApplianceBuilder(Appliance appliance) {
    appliance.setScalabilityInfo(1, 1, 99, Integer.MAX_VALUE);
    appliance.addResourceRequirement(ResourceEnum.MEMORY_MB, 256);
    appliance.addResourceRequirement(ResourceEnum.NUMBER_CPUS, 1);
  }
}
```

The interesting parts for Progress are in the `initializeProgress()` method. It contains all of the logic to build the progress tree as described above. It starts by creating a ProgressManager instance that represents the top-level of progress. It then sets the ProgressResourceBundle that contains the list of events for the top-level progress events.

The next step is to create a child ProgressManager for the Config Parsing tree, using the key from the top-level ProgressResourceBundle. This attaches this child ProgressManager to that top-level Progress event. Since the Config Parsing child ProgressManager is handling Ad-Hoc progress event, it calls the registerAdHoc() message passing in the ListResourceBundle that contains the Progress events it will send. This also returns an AdHocMarker which will be used later when triggering these Ad-Hoc Progress events.

Next up is creating the child ProgressManager for the Config Processing tree. Since this child ProgressManager is a Structured ProgressManager, it calls `registerBundle()` and not `registerAdHoc()`. You must make sure that the key's for all levels of the progress tree are unique. Having duplicates can cause unexpected consequences.

The last step in the process is generating a ProgressProxy after the entire Progress tree has been configured. The ProgressProxy is then used to trigger Progress Events, both Structured and Ad-Hoc. However, for Ad-Hoc you must pass in the AdHocMarker instance so that the ProgressProxy knows where in the tree the event needs to be added.

Export the project and deploy it to your ORACLE_HOME. Then run './abctl introspectSamplePlugin'. The result appears similar to the following:

```
> ./abctl introspectSamplePlugin
Launching introspection of component 'SamplePlugin' ...
  Step 1 of 6: dehydration starting
  Step 2 of 6: OUI processing completed
  Step 3 of 6: OPMN processing completed
    Step 1: processing file file.1
    Step 2: processing file file.2
    Step 3: processing file file.3
    Step 4: processing file file.4
    Step 1 of 4: step 1 completed
    Step 2 of 4: step 2 completed
    Step 3 of 4: step 3 completed
    Step 4 of 4: step 4 completed
  Step 6 of 6: dehydration complete
Task is done: DehydrateJob completed
Introspection complete
Storing result in catalog:
'/Users/cooluser/Documents/Perforce/my-depot/drm/src/dist/ab_home/catalog' ...
Introspection stored as 'SamplePlugin-1276728646099' in the catalog
```

That concludes the Tutorial on Progress. In this tutorial, we covered initializing a Progress tree using both Structured and Ad-Hoc progress, Child ProgressManagers, and triggering ProgressMessages. Users of your plug-in have the information they need to know that the execution of the plug-in is progressing as expected.

6.6 Tutorial 5: Job Parameters

The introspector plug-in may collect additional information from the user during its invocation, through parameters specified in the `abctl` command-line or text field entries in Studio. When the introspector plug-in registers itself, it can provide a list of Job Parameters to the framework, which the framework takes as its cue to collect the specified information from the user.

This example of the sample plug-in configures two Job Parameters, one that is a generic string and one that is a password:

```
 @Override
  public List<JobParameter> getJobParameters() {
    JobParameter jobParameter;
    List<JobParameter> jobParameters = new ArrayList<JobParameter>();

    jobParameter = jobParameterBuilder.setName(Resources.STRING_PARAM_NAME)
        .setValueType(ValueType.NAME).setRequired(false)
        .setDefaultValue(Resources.STRING_PARAM_DESC).build();
    jobParameters.add(jobParameter);

    jobParameter = jobParameterBuilder.setName(Resources.PASSWORD_PARAM_NAME)
        .setValueType(ValueType.PASSWORD).setRequired(false)
        .setDescription(Resources.PASSWORD_PARAM_DESC).build();
    jobParameters.add(jobParameter);

    return jobParameters;
  }
```

Note that in previous samples this method was returning an empty Job Parameter list. The Resources class has been created just to contain the strings being used here for parameter names and descriptions:

```
package sample.plugin.resources;

public class Resources {

  public static final String STRING_PARAM_NAME = "stringParam";
  public static final String STRING_PARAM_DESC = "just a generic string
parameter";

  public static final String PASSWORD_PARAM_NAME = "samplePassword";
  // This ends up being the password prompt:
  public static final String PASSWORD_PARAM_DESC = "Administrator password";
}
```

These parameters may be given by the user on the abctl command-line (and must be given if they are marked as being required). They are automatically shown in the abctl help for the plug-in, along with their descriptions. In Oracle Virtual Assembly Builder Studio, the introspection wizard gives you the opportunity to enter parameter values in text-entry boxes, and allows you to proceed to the next step of the introspection wizard only after values have been entered for all required parameters.

The Oracle Virtual Assembly Builder framework generates an abbreviation of the parameter names for use in abctl. For example the parameter named "samplePassword" can be specified using either "abctl introspectSamplePlugin -samplePassword" or "abctl introspectSamplePlugin -sp". The framework picks parameter abbreviations by examining the camelCase long parameter names, and complains if there are duplicate abbreviations.

The password-type job parameter is somewhat different from parameters of other types in that due to security considerations, passwords may not be entered as command-line arguments. (If they were given in plain-text on the command line they would be visible through the Unix "ps" command.) Instead, the framework prompts you for the password before dehydration begins; the short description is used as the password prompt. Also due to security considerations, passwords are kept as arrays of char rather than as String; they should not be converted to String and the character array should be cleared after the password is no longer needed.

You can retrieve the user-specified parameter values during dehydration as follows:

```
Map<String, JobParameter> jobParameterMap = context.getJobParameterMap();
JobParameter stringParameter = jobParameterMap.get(Resources.STRING_PARAM_NAME);
String stringParameterValue = stringParameter.getValue();
```

The framework provides several data types (NAME, ADDRESS, PATH, NUMERIC, BOOLEAN, PASSWORD) for parameters, but with the exception of PASSWORD, the parameter values are given to the plug-in as strings -- so for instance in the case of a BOOLEAN parameter, the plug-in must convert the strings "true" and "false" to actual boolean values.

Boolean parameters have a couple of other notable unique aspects: the abctl command creates these as optional flags, such that if the flag is specified the parameter value is "true". Therefore it doesn't make sense to have a required boolean parameter, and there's no need to specify a default value -- the default is always "false".

6.7 Sample Plug-in

A sample plug-in is included along with the tutorials. This sample has examples of many of the SDK features not covered in the tutorials. It can be built and installed just like the tutorials and will produce a deployable appliance if used for introspection.

6.8 Sample Plug-in Extension

A sample plug-in extension is also provided for reference. This sample extends the full sample plug-in. This sample shows how to access the appliance that was already created by the base plug-in and how to update a file set definition created by the base plug-in, among other things.

7

Introspection Plug-in Module Reference

This chapter describes the modules of the Oracle Virtual Assembly Builder introspection framework. This chapter contains the following sections:

7.1 Introduction

The Oracle Virtual Assembly Builder framework is made up of a number of modules. At this time, the term module is synonymous with a JAR file, but we use the term 'module' in this document. Rather than place all of the functionality in a single module, each logical grouping of functionality is in its own module. This allows you to view dependencies between each module as well as increase flexibility for deployment of the functionality in different use cases.

The goal of this section is to give a brief description of the modules that are provided in the Oracle Virtual Assembly Builder distribution. This is not a comprehensive list of all modules, only the modules that are relevant to plug-ins.

7.2 External Common [oracle.as.assemblybuilder.external.common]

The external common module contains functionality that will be used by every Plug-In. This module contains a logger, resource bundles (used by the logger for internationalization), base exceptions, and some miscellaneous classes.

7.2.1 Logging

For logging within a plug-in, use the LoggerFactory to create a Logger instance. This logger instance ensures that all log messages are placed in a common log file and with consistent formatting. This logger instance also supports OJDL for properly handling diagnostic information as well as internationalization which is required by all Oracle products. Use of the LoggerFactory and logging in general is discussed in more detail in the 'Developing an Introspector Plug-In' section.

7.2.2 Exceptions

When creating new exception types within a plug-in, they must extend either `AbException` or `AbCheckedException`. `AbException` is a `RuntimeException` and `AbCheckedException` is an `Exception`. These exception types support OJDL which is a requirement of all Oracle products.

7.2.3 Resources

Both the logging and exception classes utilize resource bundles for internationalization. By de-fault, you need to extend the ResourceBundle class in this module for the logger to access your resources. However, there are other ways to use the logger and exceptions if you choose to store your messages in another way. This is discussed in detail in Chapter 6, "Developing an Introspection Plug-in".

7.3 Progress [oracle.as.assemblybuilder.progress]

The progress module contains functionality that allows a plug-in to report progress back to the user interfaces. The progress mechanism currently in place supports both predefined progress and ad-hoc progress. Using predefined progress allows the plug-in to define a list of steps that will occur during dehydration (and eventually rehydration) such that a list can be presented to a user to see which step they are currently on and how many steps are remaining. Ad-hoc progress allows the Plug-In to provide dynamically generated text back to the user that can not be determined until run-time. This is covered in detail in Chapter 6, "Developing an Introspection Plug-in".

7.4 Introspector [oracle.as.assemblybuilder.introspector.api]

The Introspector module contains the introspector framework and the metadata API. The Introspector module registers an IntrospectorManager service, that clients use to run introspections. It also tracks all plug-in service registrations in order to dynamically discover all available plug-ins.

This module contains all of the interfaces required to build a plug-in, so your plug-in must depend on this module.

As stated in other parts of this document, the catalog is where the metadata is stored and it is the key result of the dehydration process. The introspector module exposes a subset of the catalog API through its own metadata API. The metadata API is located in this module in the 'oracle.as.assemblybuilder.introspector.metadata' package.

7.5 Appliance [oracle.as.assemblybuilder.introspector.metadata.appliance.Appliance]

An appliance is used to capture metadata for a specific installation of a server for a given product. An appliance has no children.

The following are the parts of an appliance:

CaptureId

The captureId is a unique key generated automatically when a new Appliance is created. It is used to correlate other Catalog artifacts back to the metadata.

ScalabilityInfo

Scalability info describes the minimum, maximum, target, and absolute maximum values for deploying the Appliance. Minimum is the least number of instances of this Appliance that can be deployed. Maximum is the largest number of instances of this Appliance that can be deployed. Target is the actual number of instances of this Appliance that will get created when it is deployed. Target may not be below minimum nor above maximum. Absolute maximum is the largest that maximum can be set to and is only set during dehydration, the user may not change this value at edit time. All of the other values may be set by the user at editing time.

Let's take an example where minimum is set to 1, maximum is set to 10, and target is set to 5. At initial deployment, 5 instances of the Appliance will be created. However, it will be possible for the user to add an additional 5 appliances at a later time if they decide to scale that Appliance.

Resource Requirements

Resource requirements let you define the system resources required by your appliance. Current resource requirement types: CPU_MHZ, MEMORY_MB, and NUMBER_CPUS. During deployment, these values will be used to ensure that the virtual machine where this appliance is deployed will meet or exceed the defined values.

Included Components

A list of the components that are included in this Appliance. In general, this will be the same as the Plug-In's name, and will only be a single item.

Type

The type of appliance. This can be set by the Plug-In. If not set by the Plug-In it defaults to the Plug-In name.

Inputs

An Input is by definition a Socket that is listening on some Port using some set of Protocols. You must provide the Port and Protocol values when creating an Input.

Outputs

An Output is a connection the product makes to some other Input. You must specify the Protocol used for the connection. You may also add Properties and System Properties to an Output if you need additional information on the Output at rehydration time. These properties are not used by OVAB directly and are simply passed along with the rest of the metadata. The difference between Properties and System Properties are that Properties are editable by the user, System Properties are not.

User Properties

Properties allow the plug-in to attach arbitrary name/value pairs to an appliance for use in rehydration. User properties are editable by the user.

System Properties

System Properties allow the plug-in to attach arbitrary name/value pairs to an appliance for use in rehydration. System Properties are not user editable.

Content Resources

A Content Resource can be thought of as a file. This allows you to attach files to the Appliance for transfer to the Deployed System. The idea is not to grab all files you need, but just what you need to modify in some way on its way from the Golden System to the Deployed System. For instance, configuration files may be converted from their original form into velocity templates. These templates will then be used during Rehydration to add back in the new Input/Output, Property, and System Property information. These converted files will then be put back into place over the original configuration files. You may also wish to generate a script during dehydration time that can then be added as a Content Resource and later executed at Rehydration time.

Note: Content Resources are not meant as a way to transfer files from the reference system to the deployed system. Package Definitions are used to identify sets of files to be Packaged and later moved to the deployed system.

Version

This is used to mark the metadata with a number to indicate its version. This is not a product version, but a marker. For instance, your initial release would start with version 1. If you later decided that you needed additional metadata, you would change this version to be 2. This way, when your Plug-In looks at the metadata, it can first check the version number to know what elements it should expect to exist and change its processing appropriately. The idea is that you can use a later version of your Plug-In to handle data that was generated from a previous release of the same Plug-In.

7.6 Assembly [oracle.as.assemblybuilder.introspector.metadata.assembly.Assembly]

An Assembly is a container for Appliances and Sub-Assemblies. Assemblies have Inputs, Outputs, Properties, System Properties, Content Resources, and Version the same as an Appliance. The key difference between an Appliance and an Assembly is that an Assembly is a container of other Appliances and Assemblies.

7.7 Catalog [oracle.as.assemblybuilder.catalog.api]

The catalog module is the data model for containing all of the data relating to OVAB. Think of a catalog as a hierarchical structure where data from all phases of OVAB are stored. This data consists of metadata, content resources, package definitions, packages, templates, and deployment plans.

Metadata is where Assemblies and Appliances are persisted. Metadata originates with the introspection process and is the output of a Plug-In. This data may later be edited for deployment, but it originates from the Plug-Ins.

Packages are where the binary data collected during bundling are located. Packaging is the process that occurs directly after Dehydration and is the precursor to Template

Generation. Packaging is covered in more detail in the Template Generation section below.

Templates are where deployable VM images are located. Templates are generated as a result of the Template Generation process and are used as the input into the Deployment process. Both of these are discussed in more detail in the Template Generation and Deployer sections below.

7.8 Introspector Plug-In Util [oracle.as.assemblybuilder.introspector.plugin.util]

The Introspector Plug-In Util module provides a set of utilities for plug-in development. This module defines a SPIF service 'oracle.as.assemblybuilder.introspector.plugin.util.UtilFactory' for creating utility instances. See the Javadoc documentation for a full description of each utility provided by this factory class. Of particular note is the RehydrateUtil utility class which you can use during rehydration to copy files, change their permissions, run commands as a specific user, or run velocity on templates.

8 Sample Application: Web Service API

The following sections describes a sample application using the Web Service APIs:

- Section 8.1, "Sample Application"

8.1 Sample Application

The following sample application shows how to perform each of the steps to deploy an assembly archive for a generic Oracle VM 3.0 platform.

```
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.KeyStore;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.Hashtable;
import java.util.Map;
import java.util.HashMap;
import java.util.LinkedList;
import javax.net.SocketFactory;
import javax.net.ssl.*;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

public class OvabSamples
{

  private static final String OVM_URL = "https://myovm:7001";
  private static final String OVM_POOL_NAME = "mypool";
  private static final String OVM_ADMIN_NAME = "admin";
  private static final String OVM_ADMIN_PWD = "your ovm admin password";
  private static final String TARGET_NAME = "ovm-target";

  private static final String KEYSTORE_FILE = "/ab_home/ab_
instance/config/deployer/DeployerTrust.jks";
  private static final String TRUSTSTORE_FILE = "/ab_home/ab_
instance/config/deployer/DeployerTrust.jks";
  private static final String TRUSTSTORE_PASSWORD = "your truststore password";

  private static final String ASSEMBLY_NAME = "MyAssembly";
  private static final String OVA_FILE="/archives/BaseWLS.ova";
  private static final String PLAN_FILE="/plans/BaseWLSQAPlan.xml";

  private static final String DEPLOYER_URL = "https://localhost:7002";
  private static final String CLOUD_ADMIN_USERNAME="cloudAdmin";
  private static final String CLOUD_ADMIN_PASSWORD="cloud admin password";
  private static final String APPLICATION_ADMIN_USERNAME="applicationAdmin";
  private static final String APPLICATION_ADMIN_PASSWORD="application
administrator";

  /*
   * This example shows how to deploy an OVA end-to-end on the standard OVAB
Deployer.
   */
  public static void main(String[] args) throws Exception {

    createOvmTarget();
    addOvmTargetUser();
    uploadAssemblyArchive();

    int version = registerAssemblyArchive();
    if (version >= 0) {
      String assemblyInstanceId = createAssemblyInstance(version);
      if (deployAssemblyInstance(assemblyInstanceId)) {
        message("Creation of Assembly Instance Succeeded");
      } else {
        message("Creation of Assembly Instance Failed");
      }
    } else {
      message("Registration Failed");
    }

  }

  /*
   * When using the generic Deployer, you need to define a target.
   * The target includes configuration for the backend system. In
   * this example, the backend system is Oracle VM 3.2.
   */
  public static void createOvmTarget() throws Exception {

    message("CREATE Oracle VM target "+TARGET_NAME);

    // Set up connection
    HttpURLConnection conn = getConnection(DEPLOYER_URL+"/ovab/admin");
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    String params =
      "Action=CreateTarget" +
      "&target=" + TARGET_NAME +
      "&type=ovm" +
      "&default=false" +
      "&ovm.poolName=" + OVM_POOL_NAME +
      "&ovm.vmOperationTimeout=600000" +
      "&ovm.vmmversion=3.0" +
      "&ovm.user=" + OVM_ADMIN_NAME +
      "&ovm.url=" + OVM_URL +
      "&ovm.pwd=" + OVM_ADMIN_PWD;
    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    conn.setRequestProperty("Content-Length", "" + params.length());

    // Use Cloud Admin Credentials
    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedCredentials =
      enc.encode(new String(CLOUD_ADMIN_USERNAME + ":" + CLOUD_ADMIN_
PASSWORD).getBytes());
    conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

    // Make the request
    try {
      conn.connect();
      // send the parameters
      OutputStream os = conn.getOutputStream();
      os.write(params.getBytes("UTF-8"));
      os.close();

      // Read the response
      Map<String, String> info = handleSuccessResponse(conn);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
      message("Create result: "+info);
      message("SUCCESS");

    } catch (Exception e) {
      Map<String, String> info = handleException(conn, e);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
    }

  }

  /*
   * For Oracle VM 3, users must be authorized by the administrator to use a
target
   */
  public static void addOvmTargetUser() throws Exception {

    message("ADD Oracle VM target user "+TARGET_NAME);

    // Set up connection
    HttpURLConnection conn = getConnection(DEPLOYER_URL+"/ovab/admin");
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    String params =
      "Action=AddTargetUser"+
      "&user=" + APPLICATION_ADMIN_USERNAME +
      "&target=" + TARGET_NAME;
    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    conn.setRequestProperty("Content-Length", "" + params.length());

    // Use Admin credentials
    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedCredentials =
      enc.encode(new String(CLOUD_ADMIN_USERNAME + ":" + CLOUD_ADMIN_
PASSWORD).getBytes());
```

```
    conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

    // Make the request
    try {
      conn.connect();
      // send the parameters
      OutputStream os = conn.getOutputStream();
      os.write(params.getBytes("UTF-8"));
      os.close();
      // Read the response
      Map<String, String> info = handleSuccessResponse(conn);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
      message("SUCCESS");

    } catch (Exception e) {
      Map<String, String> info = handleException(conn, e);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
    }

  }


  /*
   * This is an example of how to upload an OVA from the client to the Deployer
repository.
   */
  public static void uploadAssemblyArchive() throws Exception {
    message("UPLOAD OVA");
    // Set up connection
    HttpURLConnection conn =
      getConnection(DEPLOYER_URL+
                    "/ovab/deployer"+
                    "?Action=UploadAssemblyArchive"+
                    "&assembly.name="+ASSEMBLY_NAME+
                    "&assembly.desc="+"my sample assembly");

    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/octet-stream");
    conn.setChunkedStreamingMode(0);

    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedCredentials =
      enc.encode(new String(APPLICATION_ADMIN_USERNAME + ":" + APPLICATION_ADMIN_
PASSWORD).getBytes());
    conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

    // Make the request
    try {
      conn.connect();

      int bytesRead, bufferSize;
      int maxBufferSize = 1 * 1024 * 1024;
      byte[] buffer;
      File f = new File(OVA_FILE);
      FileInputStream fis = new FileInputStream(f);
      bufferSize = Math.min(fis.available(), maxBufferSize);
      buffer = new byte[bufferSize];
      bytesRead = fis.read(buffer, 0, bufferSize);

      OutputStream os = conn.getOutputStream();
      while (bytesRead > 0) {
        os.write(buffer, 0, bufferSize);
        bufferSize = Math.min(fis.available(), maxBufferSize);
        bytesRead = fis.read(buffer, 0, bufferSize);
      }
      fis.close();
      os.flush();
      os.close();

      // Read the response
      Map<String, String> info = handleSuccessResponse(conn);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
      message("SUCCESS");

    } catch (Exception e) {
      Map<String, String> info = handleException(conn, e);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
    }


  }

  /*
   * This is an example of how to register an assembly archive from the Deployer
   * repository to the backend system
   *
   * Registration is an asynchronous task, so this method calls
   * another method to wait for the asynchronous request to complete
   * before returning.
   */
  public static int registerAssemblyArchive() throws Exception {

    message("REGISTER Assembly Archive");

    // Set up connection
    String params =
      "Action=RegisterAssemblyArchive"+
      "&assembly.name="+ASSEMBLY_NAME+
      "&target="+TARGET_NAME;
    HttpURLConnection conn = getConnection(DEPLOYER_URL+"/ovab/deployer?"+params);
    conn.setDoInput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    conn.setRequestProperty("Content-Length", "" + params.length());

    // Use Application Admin Credentials
    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedCredentials =
      enc.encode(new String(APPLICATION_ADMIN_USERNAME + ":" + APPLICATION_ADMIN_
PASSWORD).getBytes());
```

```
    conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

    // Make the request
    try {
      conn.connect();

      // Read the response
      message("    Async Registration Request Initiated");
      Map<String, String> info = handleSuccessResponse(conn);
      String requestId =  info.get("RegisterAssemblyArchiveResult.requestId");

      // Wait for the request to complete
      if (waitForRequest(requestId)) {
        // return the version registered. This is used later for creating the
deployment
        int version =
Integer.parseInt(info.get("RegisterAssemblyArchiveResult.version"));
        return version;
      } else {
        return -1;
      }

    } catch (Exception e) {
      Map<String, String> info = handleException(conn, e);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
      return -1;
    }

  }

  /*
   * This method shows how to wait for the completion of an asynchronous Deployer
request.
   * This is used by the examples for registration and deployment of an assembly
archive.
   */
  public static boolean waitForRequest(String id) throws Exception {

    String status = "RUNNING";

    while ("RUNNING".equals(status)) {
      message("    Async request "+id+" is still in progress");

      // Set up connection
      String params =
        "Action=DescribeRequests"+
        "&request.id="+id;

      HttpURLConnection conn = getConnection(DEPLOYER_
URL+"/ovab/deployer?"+params);
      conn.setDoInput(true);
      conn.setRequestMethod("GET");
      conn.setRequestProperty("Content-Type",
"application/x-www-form-urlencoded");
      conn.setRequestProperty("Content-Length", "" + params.length());

      // Use Application Admin Credentials
      sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
      String encodedCredentials =
        enc.encode(new String(APPLICATION_ADMIN_USERNAME + ":" + APPLICATION_
ADMIN_PASSWORD).getBytes());
      conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

      // Make the request
      try {
        conn.connect();

        // Read the response
        Map<String, String> info = handleSuccessResponse(conn);
        status = info.get("DescribeRequestsResult.requestInfo.requestStatus");

        try { Thread.sleep(30*1000); } catch (InterruptedException e) { }

      } catch (Exception e) {
        Map<String, String> info = handleException(conn, e);
        for (Map.Entry<String, String> entry : info.entrySet()) {
          message("    "+entry.getKey()+"="+entry.getValue());
        }
        return false;
      }
    }

    message("    Request status is "+status);
    if ("SUCCEEDED".equals(status)) {
      return true;
    } else {
      return false;
    }
  }

  /*
   * This is an example of creating an assembly instance. This is a POST
   * operation because the deployment plan is supplied in this step.
   */
  public static String createAssemblyInstance(int version) throws Exception {
    message("CREATE Deployment");
    // Set up connection

    String urlStr =
      DEPLOYER_URL+
      "/ovab/deployer?"+
      "Action=CreateAssemblyInstance&assembly.name="+ASSEMBLY_NAME+
      "&assembly.version="+version+
      "&target="+TARGET_NAME;

    HttpURLConnection conn =
      getConnection(urlStr);

    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/octet-stream");
    conn.setChunkedStreamingMode(0);

    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedCredentials =
      enc.encode(new String(APPLICATION_ADMIN_USERNAME + ":" + APPLICATION_ADMIN_
PASSWORD).getBytes());
```

```
    conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

    // Make the request
    try {
      conn.connect();

      // Read plan document from a file and write to the web service
      int bytesRead, bufferSize;
      int maxBufferSize = 1 * 1024 * 1024;
      byte[] buffer;
      File f = new File(PLAN_FILE);
      FileInputStream fis = new FileInputStream(f);
      bufferSize = Math.min(fis.available(), maxBufferSize);
      buffer = new byte[bufferSize];
      bytesRead = fis.read(buffer, 0, bufferSize);
      OutputStream os = conn.getOutputStream();
      while (bytesRead > 0) {
        os.write(buffer, 0, bufferSize);
        bufferSize = Math.min(fis.available(), maxBufferSize);
        bytesRead = fis.read(buffer, 0, bufferSize);
      }
      fis.close();
      os.flush();
      os.close();

      // Read the response
      Map<String, String> info = handleSuccessResponse(conn);
      message("Deployment Created");
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }

      String assemblyInstanceId =
info.get("CreateAssemblyInstanceResult.assemblyInstanceId");
      return assemblyInstanceId;

    } catch (Exception e) {
      Map<String, String> info = handleException(conn, e);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }

    }

    return null;

  }

  /*
   * This is an example of how to deploy an Assembly Instance
   *
   * Deployment is an asynchronous task, so this method calls
   * another method to wait for the asynchronous request to complete
   * before returning.
   *
   */
  public static boolean deployAssemblyInstance(String assemblyInstanceId) throws
Exception {

    message("Deploy Assembly Instance");
    // Set up connection
    String params =
      "Action=DeployAssemblyInstance"+
      "&assembly.instance.id="+assemblyInstanceId;
    HttpURLConnection conn = getConnection(DEPLOYER_URL+"/ovab/deployer?"+params);
    conn.setDoInput(true);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    conn.setRequestProperty("Content-Length", "" + params.length());

    // Use Application Admin Credentials
    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedCredentials =
      enc.encode(new String(APPLICATION_ADMIN_USERNAME + ":" + APPLICATION_ADMIN_
PASSWORD).getBytes());
    conn.setRequestProperty("Authorization", "Basic " + encodedCredentials);

    // Make the request
    try {
      conn.connect();

      // Read the response
      message("    Async Deployment Request Initiated");
      Map<String, String> info = handleSuccessResponse(conn);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }

      // Wait for the deployment to complete
      String requestId = info.get("DeployAssemblyInstanceResult.requestId");
      return waitForRequest(requestId);

    } catch (Exception e) {
      Map<String, String> info = handleException(conn, e);
      for (Map.Entry<String, String> entry : info.entrySet()) {
        message("    "+entry.getKey()+"="+entry.getValue());
      }
      return false;
    }

  }


  /*
   * This is a convenient method for creating an SSL connection to the Deployer
   */
  public static HttpURLConnection getConnection(String urlString) throws Exception
{
    URL url = new URL(urlString);

    if (false) {
      return (HttpURLConnection) url.openConnection();
    }

    // Need to setup SSL connection
    SocketFactory sf = null;
    final String ALGORITHM = "sunx509";
    // For Testing
    // For testing
    String pw = TRUSTSTORE_PASSWORD;
    final char[] keystorePass = pw.toCharArray();
    final char[] truststorePass = pw.toCharArray();

    // create the private keyStore
    KeyStore ksPrivate;
    ksPrivate = KeyStore.getInstance("JKS");
    ksPrivate.load(new FileInputStream(KEYSTORE_FILE), keystorePass);

    // create the factory for the private key
    KeyManagerFactory keyManagerFactory;
    keyManagerFactory = KeyManagerFactory.getInstance(ALGORITHM);
    keyManagerFactory.init(ksPrivate, keystorePass);

    // create now the trusted keyStore
    KeyStore ksTrusted;
    ksTrusted = KeyStore.getInstance("JKS");
    ksTrusted.load(new FileInputStream(TRUSTSTORE_FILE), truststorePass);

    // create a trust manager factory using the same algorithm.
    TrustManagerFactory trustManagerFactory;
    trustManagerFactory = TrustManagerFactory.getInstance(ALGORITHM);
    trustManagerFactory.init(ksTrusted);

    SSLContext sslc = SSLContext.getInstance("TLS");
    sslc.init(keyManagerFactory.getKeyManagers(),
trustManagerFactory.getTrustManagers(), null);

    sf = sslc.getSocketFactory();
    HttpsURLConnection httpsURLConn = (HttpsURLConnection)url.openConnection();
    httpsURLConn.setSSLSocketFactory((SSLSocketFactory) sf);

    httpsURLConn.setHostnameVerifier(new HostnameVerifier() {
        public boolean verify(String host, SSLSession sslsession) {
          return true;
        }
      });
    return (HttpURLConnection)httpsURLConn;

  }

  /*
   * This is a convenience method for extracting the information in the
   * response from a failed Deployer Web service call.
   */
  public static Map<String,String> handleException(HttpURLConnection conn,
Exception e) throws IOException, SAXException {

    Map<String,String> result = new HashMap<String,String>();

    message("ERROR: "+e.getMessage());
    message("RESPONSE CODE: "+conn.getResponseCode());
    if (conn.getResponseCode() == (HttpURLConnection.HTTP_NOT_FOUND)) {
      message("Invalid request, web service not found");
      return result;
    } else if (conn.getResponseCode() == (HttpURLConnection.HTTP_UNAUTHORIZED)) {
      message("User is not authorized to perform this action");
      return result;
    } else if (conn.getResponseCode() == (HttpURLConnection.HTTP_FORBIDDEN)) {
      message("User is not authorized to perform this action");
```

```
        return result;
      }

      // Read the response
      DataInputStream input = new DataInputStream(conn.getErrorStream());
      String str;
      StringBuffer response = new StringBuffer();
      try {
        while (null != ((str = input.readLine()))) {
          response.append(str);
        }
        input.close();

        // Parse the response
        XMLReader reader = XMLReaderFactory.createXMLReader();
        reader.setContentHandler(new ResponseHandler(result));
        reader.parse(new InputSource(new StringReader(response.toString())));

      } catch (Exception e2) {
        message("Failure reading error stream: "+e2);
      }

      return result;
    }

    /*
     * This is a convenience method for extracting the information in the
     * response from a successful Deployer Web service call.
     */
    public static Map<String,String> handleSuccessResponse(HttpURLConnection conn)
  throws IOException, SAXException {
      HashMap<String,String> result = new HashMap<String,String>();
      DataInputStream input = new DataInputStream(conn.getInputStream());
      String str;
      StringBuffer response = new StringBuffer();
      while (null != ((str = input.readLine()))) {
        response.append(str);
      }
      input.close();

      // Parse the response
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(new ResponseHandler(result));
      reader.parse(new InputSource(new StringReader(response.toString())));

      return result;
    }

    /*
     * The responses returned from the Deployer Web service are XML
     * documents. This handler translates the structure of the document
     * into a Map which is useful for picking out certain element.  It
     * also prints the entries to System.out.
     */
    public static class ResponseHandler extends DefaultHandler {

      private Map<String,String> info;

      public ResponseHandler(Map<String,String> info) {
        this.info = info;
      }

      LinkedList<String> elements = new LinkedList<String>();

      public void startElement(String namespaceURI, String elementName, String
  qName, Attributes attrs) {
        elements.add(elementName);
      }

      public void characters(char ch[], int start, int length) {
        String str = new String(ch,start,length);

        //message(String.format("    %s=%s", getCurrentKey(), str));
        info.put(getCurrentKey(), str);
      }

      private String getCurrentKey() {
        String result = null;
        for (String s : elements) {
          if (result != null) {
            result = result + "." +s;
          } else {
            result = s;
          }
        }
        return result;
      }

      public void endElement(String namespaceURI, String elementName, String qName)
  {
        elements.removeLast();
      }
    }

    private static void message(String msg) {
      System.out.println("["+new Date(System.currentTimeMillis())+"] "+msg);
   }

  }
```

A Web Service Schema

This appendix defines the schema for the response information returned by the Deployer Web service. The types defined in this schema are referenced by the "Response Content" descriptions in earlier chapters of the guide.

- Section A.1, "Schema"

A.1 Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oracle.com/ovab/deployer/webservice/bindings"
    xmlns:tns="http://www.oracle.com/ovab/deployer/webservice/bindings"
    elementFormDefault="qualified">
    <xsd:element name="Path">
        <xsd:complexType>
            <xsd:attribute name="value" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="InstancePath">
        <xsd:complexType>
            <xsd:attribute name="value" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="AppliancePath">
        <xsd:complexType>
            <xsd:attribute name="value" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Paths">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="tns:Path" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="AssemblyInstanceId">
        <xsd:complexType>
            <xsd:attribute name="value" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="AssemblyInstanceState">
        <xsd:complexType>
            <xsd:attribute name="id" type="xsd:string"/>
            <xsd:attribute name="state" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UploadAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CopyAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CanUploadAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="canUpload"
type="tns:canUploadType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="versions"
type="tns:VersionsType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="ErrorDetail">
        <xsd:sequence>
            <xsd:element name="code" minOccurs="1" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string"/>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="message" minOccurs="1" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string"/>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="cause" minOccurs="0" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string"/>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="action" minOccurs="0" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string"/>
                </xsd:simpleType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="ErrorResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="type">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string"/>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="error-detail" type="tns:ErrorDetail"
minOccurs="1" maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="VersionsType">
        <xsd:sequence>
            <xsd:element name="version" maxOccurs="unbounded"
type="tns:versionType"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="RegisterAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
                <xsd:element name="targetName"
type="tns:targetNameType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UnregisterAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
                <xsd:element name="targetName"
type="tns:targetNameType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="requestIdType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="nameType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="canUploadType">
        <xsd:restriction base="xsd:boolean"/>
    </xsd:simpleType>
    <xsd:simpleType name="versionType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="patchIdType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="payloadNameType">
```

```
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="applianceInstanceIdType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:element name="DescribeAssemblyArchivesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assembly" type="tns:Assembly" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="AssemblyVersion">
        <xsd:sequence>
            <xsd:element name="version" type="tns:versionType"></xsd:element>
            <xsd:element name="description"
type="tns:descriptionType"></xsd:element>
            <xsd:element name="appliances">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="appliance" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="plans">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="plan" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="versionHistory">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="version" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="appliedPatches">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="patchId" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Assembly">
        <xsd:sequence>
            <xsd:element name="name" type="tns:nameType"></xsd:element>
            <xsd:element name="owner" type="xsd:boolean"></xsd:element>
            <xsd:element name="versions">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="version" type="tns:AssemblyVersion"
```

```
maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="plans">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="plan" maxOccurs="unbounded"
type="xsd:string"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="users">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="user" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="descriptionType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:element name="UpdateAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="assemblyInstanceIdType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:element name="CreateAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeployAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UndeployAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId" 
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId" 
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="StartAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId" 
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId" 
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="StopAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId" 
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId" 
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="instanceIdType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType name="InstancesType">
        <xsd:sequence>
            <xsd:element name="instanceId" type="tns:instanceIdType" 
maxOccurs="unbounded"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="targetNameType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType name="Instance">
        <xsd:sequence>
            <xsd:element name="assemblyInstanceId" 
type="xsd:string"></xsd:element>
            <xsd:element name="instanceId" type="xsd:string"></xsd:element>
            <xsd:element name="instanceState" type="xsd:string"></xsd:element>
            <xsd:element name="applianceId" type="xsd:string"></xsd:element>
            <xsd:element name="appliancePath" type="xsd:string"></xsd:element>
            <xsd:element name="vmName" type="xsd:string"></xsd:element>
            <xsd:element name="vmId" type="xsd:string"></xsd:element>
            <xsd:element name="ipaddresses">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="ipaddress" maxOccurs="unbounded" 
type="xsd:string"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="vnets">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="vnet" maxOccurs="unbounded"
type="tns:Vnet"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="volumes">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="volume" maxOccurs="unbounded"
type="xsd:string"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="meteringMap">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="metering" maxOccurs="unbounded">
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name="name"
type="xsd:string"></xsd:element>
                                    <xsd:element name="value"
type="xsd:long"></xsd:element>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="networkInterfaces">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="interface" maxOccurs="unbounded"
type="tns:NetworkInterface"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="AssemblyInstance">
        <xsd:sequence>
            <xsd:element name="assemblyInstanceId"
type="xsd:string"></xsd:element>
            <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
            <xsd:element name="assemblyVersion" type="xsd:string"></xsd:element>
            <xsd:element name="state" type="xsd:string"></xsd:element>
            <xsd:element name="targetName" type="xsd:string"></xsd:element>
            <xsd:element name="appliances">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="appliance" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="versionHistory">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="version" type="xsd:string"
minOccurs="0" maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeAssemblyInstancesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyInstance" type="tns:AssemblyInstance"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="RequestInfo">
        <xsd:sequence>
            <xsd:element name="requestId" type="tns:requestIdType"></xsd:element>
            <xsd:element name="requestStatus" type="xsd:string"></xsd:element>
            <xsd:element name="requestType" type="xsd:string"></xsd:element>
            <xsd:element name="operationType" type="xsd:string"></xsd:element>
            <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
            <xsd:element name="assemblyVersion" type="xsd:string"></xsd:element>
            <xsd:element name="message" type="xsd:string"></xsd:element>
            <xsd:element name="errorDetails">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="error-detail" type="tns:ErrorDetail"
minOccurs="1" maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="creationTimeStamp" type="xsd:long"></xsd:element>
            <xsd:element name="completionTimeStamp" type="xsd:long"></xsd:element>
            <xsd:element name="progressMessages">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="progressMessage" maxOccurs="unbounded"
type="xsd:string"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeRequestsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestInfo" type="tns:RequestInfo"
minOccurs="0" maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeDeployerResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="specificationVersion"
type="xsd:string"></xsd:element>
```

```
                <xsd:element name="implementationVersion"
type="xsd:string"></xsd:element>
                <xsd:element name="assemblyStoreFreeSpace"
type="xsd:long"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CreateTargetResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeTargetNamesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteTargetResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="SetDefaultTargetResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="targetName" type="xsd:string"></xsd:element>
                <xsd:element name="targetType" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeApplianceInstancesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="instance" type="tns:Instance"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="AddTargetUserResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="xsd:string"></xsd:element>
                <xsd:element name="user" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RemoveTargetUsersResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="xsd:string"></xsd:element>
                <xsd:element name="users">
                    <xsd:complexType>
                        <xsd:sequence>
```

```
                            <xsd:element name="user" type="xsd:string" 
maxOccurs="unbounded"></xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="Properties">
        <xsd:sequence>
            <xsd:element name="entry" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="key" minOccurs="1" maxOccurs="1" 
type="xsd:string"/>
                        <xsd:element name="value" minOccurs="1" maxOccurs="1" 
type="xsd:string"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="TargetUser">
        <xsd:sequence>
            <xsd:element name="userName" type="xsd:string"></xsd:element>
            <xsd:element name="targetProperties" 
type="tns:Properties"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeTargetUsersResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="user" type="tns:TargetUser" 
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeUserTargetsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="xsd:string" 
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="GetTargetTypeResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="targetType" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="GetDefaultTargetResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="targetName" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="ScalingGroup">
```

```
        <xsd:sequence>
            <xsd:element name="scalingGroupId" type="xsd:string"></xsd:element>
            <xsd:element name="min" type="xsd:int"></xsd:element>
            <xsd:element name="max" type="xsd:int"></xsd:element>
            <xsd:element name="target" type="xsd:int"></xsd:element>
            <xsd:element name="current" type="xsd:int"></xsd:element>
            <xsd:element name="initial" type="xsd:int"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeScalingGroupsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="scalingGroups" type="tns:ScalingGroup"
minOccurs="0" maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ScaleApplianceResult">
       <xsd:complexType>
        <xsd:sequence>
       <xsd:element name="requestId" type="tns:requestIdType"></xsd:element>
            <xsd:element name="scalingGroupId" type="xsd:string"></xsd:element>
            <xsd:element name="assemblyInstanceId"
type="xsd:string"></xsd:element>
        </xsd:sequence>
       </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="Vnet">
       <xsd:sequence>
           <xsd:element name="vnetId" type="xsd:string" minOccurs="0"
maxOccurs="1"></xsd:element>
           <xsd:element name="vnetName" type="xsd:string" minOccurs="0"
maxOccurs="1"></xsd:element>
           <xsd:element name="private" type="xsd:boolean" minOccurs="0"
maxOccurs="1"></xsd:element>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeVnetsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="vnet" type="tns:Vnet" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="Target">
        <xsd:sequence>
            <xsd:element name="targetName"
type="tns:targetNameType"></xsd:element>
            <xsd:element name="targetType" type="xsd:string"></xsd:element>
            <xsd:element name="targetCapability" type="xsd:string"></xsd:element>
            <xsd:element name="status" type="xsd:string"></xsd:element>
            <xsd:element name="vnets">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="vnet"  minOccurs="0"
maxOccurs="unbounded" type="tns:Vnet"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="volumeNames">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="volumeName"  minOccurs="0"
maxOccurs="unbounded" type="xsd:string"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="availabilityMap">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="metering"  minOccurs="0"
maxOccurs="unbounded">
                            <xsd:complexType>
                                <xsd:sequence>
                                    <xsd:element name="name"
type="xsd:string"></xsd:element>
                                    <xsd:element name="value"
type="xsd:long"></xsd:element>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeTargetsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="target" type="tns:Target" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="RegistrationInfo">
        <xsd:sequence>
            <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
            <xsd:element name="assemblyVersion" type="xsd:string"></xsd:element>
            <xsd:element name="targetName" type="xsd:string"></xsd:element>
            <xsd:element name="registrationId" type="xsd:string"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeRegistrationsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="registration" type="tns:RegistrationInfo"
minOccurs="0" maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="TargetConfiguration">
        <xsd:sequence>
            <xsd:element name="targetName" type="xsd:string"></xsd:element>
            <xsd:element name="targetType" type="xsd:string"></xsd:element>
            <xsd:element name="properties">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="property" maxOccurs="unbounded">
                            <xsd:complexType>
```

```
                                <xsd:sequence>
                                    <xsd:element name="name"
type="xsd:string"></xsd:element>
                                    <xsd:element name="value"
type="xsd:string"></xsd:element>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DescribeTargetConfigurationsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="targetConfiguration"
type="tns:TargetConfiguration" minOccurs="0" maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RestartAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RedeployAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteRequestsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestsPurged"
type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeAssemblyUsersResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
                <xsd:element name="users" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="AddAssemblyUsersResult">
        <xsd:complexType>
```

```
            <xsd:sequence>
                <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
                <xsd:element name="users" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RemoveAssemblyUsersResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
                <xsd:element name="users" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CreateTagsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="resources" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                <xsd:element name="tags">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="tag" maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="key"
type="xsd:string"></xsd:element>
                                        <xsd:element name="value"
type="xsd:string"></xsd:element>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteTagsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="resources" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                <xsd:element name="tags" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeTagsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="tags">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="tag" maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="key"
```

```
type="xsd:string"></xsd:element>
                                        <xsd:element name="value"
type="xsd:string"></xsd:element>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UpdateTargetResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="targetName" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteFailedApplianceInstancesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="applianceId" type="xsd:string"></xsd:element>
                <xsd:element name="applianceInstanceIds" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ValidateAssemblyInstanceResourcesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="status" type="xsd:string"></xsd:element>
                <xsd:element name="errors" maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="SuspendAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyInstanceId"
type="xsd:string"></xsd:element>
                <xsd:element name="requestId" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ResumeAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyInstanceId"
type="xsd:string"></xsd:element>
                <xsd:element name="requestId" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="NetworkInterface">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"></xsd:element>
            <xsd:element name="ipAddress" type="xsd:string"></xsd:element>
            <xsd:element name="networkName" type="xsd:string"></xsd:element>
            <xsd:element name="private" type="xsd:boolean"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="UploadDeploymentPlanResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
                <xsd:element name="plan" type="tns:nameType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteDeploymentPlanResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
                <xsd:element name="plan" type="tns:nameType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UploadAssemblyResourcesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UploadPatchResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
                <xsd:element name="patchId" type="tns:patchIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PatchAssemblyArchiveResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyName"
type="tns:nameType"></xsd:element>
                <xsd:element name="originalVersion"
type="tns:versionType"></xsd:element>
                <xsd:element name="patchedVersion"
type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PatchAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
                <xsd:element name="originalVersion"
type="tns:versionType"></xsd:element>
```

```
                <xsd:element name="patchedVersion"
type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RevAssemblyInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
                <xsd:element name="originalVersion"
type="tns:versionType"></xsd:element>
                <xsd:element name="patchedVersion"
type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PatchApplianceInstanceResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="applianceInstanceId"
type="tns:applianceInstanceIdType"></xsd:element>
                <xsd:element name="originalVersion"
type="tns:versionType"></xsd:element>
                <xsd:element name="patchedVersion"
type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UploadPayloadResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="payloadName"
type="tns:payloadNameType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="SendPayloadResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeletePayloadResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="payloadName"
type="tns:payloadNameType"></xsd:element>
                <xsd:element name="assemblyName"
type="tns:nameType"></xsd:element>
                <xsd:element name="assemblyVersion"
type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ExecuteCommandResult">
```

```
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="applianceInstanceId"
type="tns:applianceInstanceIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RollbackInstancePatchResult">
      <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="assemblyInstanceId"
type="tns:assemblyInstanceIdType"></xsd:element>
                <xsd:element name="patchId" type="tns:patchIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="RollbackArchivePatchResult">
      <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="patchId" type="tns:patchIdType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeletePatchResult">
      <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="patchId" type="tns:patchIdType"></xsd:element>
                <xsd:element name="assemblyName"
type="tns:nameType"></xsd:element>
                <xsd:element name="assemblyVersion"
type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeAssemblyInstancePatchesResult">
      <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="patch" type="tns:AssemblyPatch"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribePatchesResult">
      <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="patch" type="tns:AssemblyPatch"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribePayloadsResult">
      <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="payload" type="tns:Payload"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="SoftwarePatch">
        <xsd:sequence>
            <xsd:element name="targetAppliance" type="xsd:string"></xsd:element>
            <xsd:element name="nativeId" type="xsd:string"></xsd:element>
            <xsd:element name="format" type="xsd:string"></xsd:element>
            <xsd:element name="resolvedIssues" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="AssemblyPatch">
        <xsd:sequence>
            <xsd:element name="patchId" type="tns:patchIdType"></xsd:element>
            <xsd:element name="creationTime" type="xsd:long"></xsd:element>
            <xsd:element name="assemblyName" type="tns:nameType"></xsd:element>
            <xsd:element name="assemblyVersion"
type="tns:versionType"></xsd:element>
            <xsd:element name="softwarePatch" type="tns:SoftwarePatch"
maxOccurs="unbounded"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Payload">
        <xsd:sequence>
            <xsd:element name="payloadName"
type="tns:payloadNameType"></xsd:element>
            <xsd:element name="creationTime" type="xsd:long"></xsd:element>
            <xsd:element name="assemblyName" type="tns:nameType"></xsd:element>
            <xsd:element name="assemblyVersion"
type="tns:versionType"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DeleteAssemblyResourcesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeAssemblyResourcesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="name" type="tns:nameType"></xsd:element>
                <xsd:element name="version" type="tns:versionType"></xsd:element>
                <xsd:element name="resources">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="resource" maxOccurs="unbounded"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="ResourceType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="APPLIANCE_INSTANCE"/>
    <xsd:enumeration value="OVA"/>
    <xsd:enumeration value="DEPLOYMENT"/>
    <xsd:enumeration value="REGISTRATION"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="LogEventType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="VM_STATE"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="VMState">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CREATED"/>
            <xsd:enumeration value="STARTED"/>
            <xsd:enumeration value="STOPPED"/>
            <xsd:enumeration value="DESTROYED"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="LogEvent" abstract="true">
        <xsd:sequence>
            <xsd:element name="timeStamp" type="xsd:long"></xsd:element>
            <xsd:element name="user" type="xsd:string"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="VMStateEvent">
        <xsd:complexContent>
        <xsd:extension base="tns:LogEvent">
            <xsd:sequence>
                <xsd:element name="assemblyInstanceId"
type="xsd:string"></xsd:element>
                <xsd:element name="assemblyName" type="xsd:string"></xsd:element>
                <xsd:element name="appliancePath" type="xsd:string"></xsd:element>
                <xsd:element name="instanceIndex" type="xsd:int"></xsd:element>
                <xsd:element name="cpu" type="xsd:long"></xsd:element>
                <xsd:element name="disk" type="xsd:long"></xsd:element>
                <xsd:element name="memory" type="xsd:long"></xsd:element>
                <xsd:element name="vmState" type="tns:VMState"></xsd:element>
                <xsd:element name="eventType"
type="tns:LogEventType"></xsd:element>
            </xsd:sequence>
        </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="DescribeResourcesByTagsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="resources">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="resource" maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="resourceName"
minOccurs="1" maxOccurs="1" type="xsd:string"></xsd:element>
                                        <xsd:element name="resourceType"
minOccurs="1" maxOccurs="1" type="tns:ResourceType"></xsd:element>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DeleteLogEventsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="eventType"
type="tns:LogEventType"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeLogEventsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="events">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element minOccurs="0" maxOccurs="unbounded"
name="event" type="tns:LogEvent"></xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeApplianceInstanceMetricsResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="metrics" minOccurs="0" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="instanceId"
type="xsd:string"></xsd:element>
                            <xsd:element name="vmId"
type="xsd:string"></xsd:element>
                            <xsd:element name="metric" minOccurs="0"
maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="name"
type="xsd:string"></xsd:element>
                                        <xsd:element name="values">
                                            <xsd:complexType>
                                                <xsd:sequence>
                                                    <xsd:element name="value"
minOccurs="0" maxOccurs="unbounded" type="xsd:double"></xsd:element>
                                                </xsd:sequence>
                                            </xsd:complexType>
                                        </xsd:element>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

```
<xsd:element name="StartMetricsPollingResult">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="watchItemId" type="xsd:long"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="StopMetricsPollingResult">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="watchItemId" type="xsd:long"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="AddRuleSetResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="DeleteRuleSetResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="DeleteRuleResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="EnableRuleResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="DisableRuleResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="DisableRuleSetResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="EnableRuleSetResult">
  <xsd:complexType>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="RuleInformation">
  <xsd:sequence>
    <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
    <xsd:element name="ruleSetName" type="xsd:string"></xsd:element>
    <xsd:element name="ruleName" type="xsd:string"></xsd:element>
    <xsd:element name="status" type="xsd:string"></xsd:element>
    <xsd:element name="message" type="xsd:string"></xsd:element>
    <xsd:element name="lastTriggered" type="xsd:long"></xsd:element>
    <xsd:element name="contiguousTriggers" type="xsd:int"></xsd:element>
    </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="DescribeRulesResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="rule-information" minOccurs="0" maxOccurs="unbounded"
type="tns:RuleInformation"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="RuleMetricInformation">
      <xsd:sequence>
        <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
        <xsd:element name="ruleSetName" type="xsd:string"></xsd:element>
        <xsd:element name="metricName" type="xsd:string"></xsd:element>
        <xsd:element name="status" type="xsd:string"></xsd:element>
        <xsd:element name="message" type="xsd:string"></xsd:element>
        <xsd:element name="lastEvaluated" type="xsd:long"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="DescribeRuleMetricsResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="rule-metric-information" minOccurs="0"
maxOccurs="unbounded" type="tns:RuleMetricInformation"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="RuleActionInformation">
      <xsd:sequence>
        <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
        <xsd:element name="ruleSetName" type="xsd:string"></xsd:element>
        <xsd:element name="ruleName" type="xsd:string"></xsd:element>
        <xsd:element name="status" type="xsd:string"></xsd:element>
        <xsd:element name="message" type="xsd:string"></xsd:element>
        <xsd:element name="started" type="xsd:long"></xsd:element>
        <xsd:element name="completed" type="xsd:long"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="DescribeRuleActionsResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="rule-action-information" minOccurs="0"
maxOccurs="unbounded" type="tns:RuleActionInformation"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="CreateAssemblyInstanceSnapshotResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
          <xsd:element name="requestId" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="CreateDiskSnapshotResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="applianceInstanceId" type="xsd:string"></xsd:element>
          <xsd:element name="diskName" type="xsd:string"></xsd:element>
          <xsd:element name="requestId" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="CreateDisksSnapshotResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
          <xsd:element name="diskName" type="xsd:string"></xsd:element>
          <xsd:element name="requestId" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="CreateApplianceInstanceSnapshotResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="applianceInstanceId" type="xsd:string"></xsd:element>
          <xsd:element name="requestId" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="DeleteSnapshotsResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
            <xsd:element name="snapshotIds">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="snapshotId" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="Snapshot">
      <xsd:sequence>
        <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
        <xsd:element name="snapshotId" type="xsd:string"></xsd:element>
        <xsd:element name="snapshotName" type="xsd:string"></xsd:element>
        <xsd:element name="snapshotDescription" type="xsd:string"></xsd:element>
        <xsd:element name="creationTime" type="xsd:long"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="SnapshotDisk">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"></xsd:element>
        <xsd:element name="ovfId" type="xsd:string"></xsd:element>
        <xsd:element name="nativeDiskId" type="xsd:string"></xsd:element>
        <xsd:element name="memberId" type="xsd:string"></xsd:element>
        <xsd:element name="sharable" type="xsd:boolean"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Snapshots">
        <xsd:sequence>
            <xsd:element name="snapshot" minOccurs="0" maxOccurs="unbounded"
type="tns:Snapshot"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="SnapshotDisks">
        <xsd:sequence>
            <xsd:element name="snapshotDisk" minOccurs="0" maxOccurs="unbounded"
type="tns:SnapshotDisk"></xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="DescribeSnapshotsResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="snapshots" type="tns:Snapshots"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="RestoreSnapshotResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="assemblyInstanceId" type="xsd:string"></xsd:element>
          <xsd:element name="requestId" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="DescribeSnapshotDisksResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="snapshotDisks"
type="tns:SnapshotDisks"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="DescribeRequestDetailResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId" type="xsd:string"></xsd:element>
                <xsd:element name="status" type="xsd:string"></xsd:element>
                <xsd:element name="totalSteps" type="xsd:int"></xsd:element>
                <xsd:element name="completedSteps" type="xsd:int"></xsd:element>
                <xsd:element name="messages">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="message" maxOccurs="unbounded"
type="xsd:string"></xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="StartApplianceInstancesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="applianceInstanceIds" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="StopApplianceInstancesResult">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="requestId"
type="tns:requestIdType"></xsd:element>
                <xsd:element name="applianceInstanceIds" type="xsd:string"
maxOccurs="unbounded"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```