Agility Platform

Services SDK 1.0

Agility Platform 10.0

Published: September 2015

UNITED STATES GOVERNMENT LEGEND

Contents

About This Guide

This guide provides information used to build a custom cloud service adapter for use with the Agility Platform. It assumes an in-depth understanding of software development and cloud-based systems, as well as familiarity with the Agility Platform and the configuration and administration of service provider types.

> **Important:** If the Release Notes document provided with the CSC Agility Platform contains details that differ from the information in this guide, the information in the release notes supersedes the information in this guide.

> **Note:** Most Agility Platform guides are available in Adobe Acrobat Reader Portable Document Format (PDF) in the customer area of the CSC Agility Platform FTP site:
>
> ftp://ftp-aus.servicemesh.com

Related Documents

In addition to this guide, you can find useful information in the following publications:

- *Agility Platform User Guide* - This guide provides information and procedures to help you effectively use the Agility Platform and the primary web application.
- *Agility Platform Administrator Guide* - This guide provides information and procedures for administrators to set up and configure the Agility Platform for use across an organization.

- *Agility Platform Policy Development Guide* - This guide provides information and procedures for defining and applying policies to manage compute resource use and provisioning in the Agility Platform.

- *Agility Store User Guide* - This guide provides conceptual and instructional information about day-to-day use of the Agility Store and Agility Launchpad web applications.

- *Agility Store Administrator Guide* - This guide provides conceptual and instructional information about day-to-day administration of the Agility Store. It includes information about setting up and configuring the web application for a community of Agility Store users.

- *Agility Release Manager User Guide* - This guide provides information and procedures to help you effectively use the Agility Release Manager web application.

- *Agility Release Manager DevOps Project Management Guide* - This guide provides information and procedures to help administrators and project managers effectively configure the platform services, solutions, artifacts, and SDLC environments used to create and govern deployments in the Agility Release Manager web application.

- *Agility Platform Installation and Setup Guide* - This guide helps administrators install and perform an initial set up of a new Agility Platform installation.

- *Agility Platform Upgrade Guide* - This guide provides information and procedures to help administrators upgrade an existing Agility Platform server.

- *Agility Platform Tools Installation Guide* - This guide provides instructions for installing Agility Platform Tools on VM instances to make them compatible for onboarding to the Agility Platform or on custom images to import as stacks for instances managed by the Agility Platform. It also provides repackaging instructions for Agility Platform Tools and Agility Platform Solutions that are separate from their installation RPM files, DEB files, and other third-party software.

- *Agility Platform REST API Guide* - This guide provides usage information and examples to help you use the Agility Platform base REST API. It also provides syntax and reference information for the Metric and Event REST APIs.

- *Agility Platform Scripting API Guide* - This guide provides information, syntax, and reference tables to help you use the Scripting API.

- *Agility Platform Cloud SDK* - This guide provides information about using the Cloud SDK to build custom cloud provider adapters for the Agility Platform Platform. Administrators can make additional cloud provider types available and configure the cloud providers for Agility Platform users.

- *Agility Platform Help* - This Help system provides comprehensive information and procedures for the Agility Platform web application.

You can access the Agility Platform product documentation in the following ways:

- The product documentation (for web applications and APIs) is available in PDF format on the server where the Agility Platform is installed. Access these PDF files in the `...agility\help` directory.

- Click the Help link at the top-right of the Agility Platform, Agility Release Manager, or Agility Store web applications to launch the Help for the features provided by that application. These browser-based WebHelp systems provide the same information in the PDF user and administrator guides in an easy-to-use format.

- All Agility Platform documentation, including installation and upgrade, is available in PDF format on the FTP client site at the following location:

  FTP:\<*YourClientSite*>\docs

  > *Note:* Client credentials are required to connect to the Agility Platform FTP site.

Conventions

The following typographic and formatting conventions are used throughout this guide to help you navigate and understand information.

Notice icons

Notes provide information of special importance or that cannot be suitably presented in the main text. The following icons call your attention to the information and its purpose:

Icon	Notice type	Description
	Information Note	Provides additional information to supplement the primary text or calls your attention to a dependency or information that can apply in special cases
	Important Note	Provides information that is essential to the successful completion of a task or to potentially undesirable side effects

| | Security Note | Provides information about the required access permissions to perform the described operations or considerations that apply to securing your systems |

| | Warning Note | Provides information that alerts you to potential loss of data or to potentially destructive side effects |

Text conventions

The following special text formatting is used throughout this guide to help you locate and interpret information:

Convention	Description
Syntax	The word *syntax* means that you must evaluate the provided syntax and supply the appropriate values for the placeholders that appear in angle brackets. *Example:* To change directory, use the following syntax: `cd <directory>` In this example, you must supply a directory for <directory>.
Commands	The word *command* means that you must enter the command exactly as shown and then press Return or Enter. Commands appear in a monospace font. *Example:* `cd <directory>`
Keyboard key names	When you must press two or more keys simultaneously, the key names are linked with a plus sign (+). *Example:* Press Ctrl+Alt+Del

Words in bold	Bold text is used for the following: • Emphasizes a point • Identifies items to select or click in the user interface, including menu names, menu commands, and button names *Examples:* At least one element **must** be a static value. In the Configure panel, select Add New. Click OK.
Words in italic	Italicized text is used for the following: • Denotes a new term where it is defined in the text • Emphasizes a particular word or term • Identifies a book title

Feedback About This Guide

Your suggestions are very important to us and help make our documentation more useful to all of our customers. Please submit comments about this document to CSC.

Include the following information in your comments:

- Document title
- Publish date
- Page number

Chapter 1

The service framework provides an integration path to any external service required by an application or the infrastructure on which it is deployed, such as the following examples:

■ Database as a service (DBaaS)

■ Messaging load balancer as a service (LBaaS)

■ IP address management (IPAM)

■ Domain name services (DNS)

In the Agility Platform, a cloud service adapter is responsible for the following functionality:

■ Describing externally hosted application services

■ Translating Agility Platform provisioning commands into service provisioning commands

■ Configuring application services

Service Framework Core Concepts

Before you begin your service adapter development project, we recommend that you familiarize yourself with the following core concepts:

Service

A *service* defines an integration with an external, cloud-based system that can be used by an application or deployment infrastructure. A defined application service can be used in the composition of applications to provide support for functionality, such as a web service, database service, or message bus service. At deployment, the service instance is provisioned within the topology.

Service provider

A *service provider* is a component whose definition, based on the service provider type, is configured by Agility Platform administrators. The service provider definition specifies all of the parameters required to communicate with a cloud-based service provider and provision a

service of that type. Upon deployment of the blueprint, the deployment policy evaluates the set of available target environments based on the availability of the required service and creates a binding from the generated deployment to the service provider.

Service connection

A *service connection* is a design asset that specifies connection settings between an application service and the design elements that consume the service. The design elements for a service connection are service-to-service or service-to-workload. At runtime, a dependent node cannot start until all dependencies have successfully started. Additionally, if a dependency (service) stops or restarts, all workload dependent nodes (VM instances) execute all reconfigure scripts included in assigned packages.

Service instance

A *service instance* is a runtime service that is created at deployment. Agility Platform creates a service binding to map the service requirements/configuration to a specific service provider. Users can use runtime operations to start/stop/restart/release the service instance.

Service adapter

A *service adapter* is an OSGi bundle comprised of Java source files and provides the following functions within the Agility Platform:

- Creates the metadata that enables the supported service and service connections for use in a blueprint
- Creates the metadata that enables the definition of the service provider
- Creates service instances for deployed blueprints
- Manages lifecycle events for service provider, service instances, and the instances and service instances connected to a service instance

Service binding

A *service binding* is created at deployment to map the service instance to a specific service provider. The binding relates the service instance to the service provider and captures the configuration of the service instance when the blueprint is deployed. The binding additionally relates the service instance to any dependent services or workloads.

Service provider type

A *service provider type* is defined by the service adapter to specify configuration parameters required to enable the service on the provider. Agility Platform incorporates the Service Provider Type asset type into its object model to support a hierarchy of defined service providers.

Service Adapters

CSC provides the Services SDK so that you can build cloud service adapters that adhere to an asynchronous model. With the asynchronous model, the processing follows two guidelines for building an asynchronous service adapter.

1. The Agility Platform sends commands as asynchronous messages. The adapter executes the command and returns a message at completion.

2. The adapter communicates with the service provider in an asynchronous manner using the Async HTTP library.

OSGi Bundles for Service Adapters

Service adapters created for the Agility Platform Services SDK are distributed and deployed as OSGi bundles. The following naming convention is used:

```
com.<vendor>.agility.adapters.service.<servicetype>.<version>.jar
```

The convention for the version number is $V_{major}.V_{minor}.V_{patch}$

The Services SDK includes the following packages.

■   `com.servicemesh.agility.sdk.service.msgs` - JAXB generated classes from xml schema definition of request/response messages

■   `com.servicemesh.agility.sdk.service.spi` - Provides abstract base class for all adapters along with interfaces that are implemented by the adapters to expose supported functionality

■   `com.servicemesh.agility.sdk.service.operations` - Default implementations of the operations/interfaces that can be sub-classed to override specific methods.

■   `com.servicemesh.agility.sdk.service.exceptions` - Base class for exceptions thrown by the adapter.

■   `com.servicemesh.agility.sdk.service.helper` - Provides helper/convenience methods for manipulating API objects

Service Adapter Dependencies

Service adapters have dependencies on the following OSGi bundles that are distributed with the Agility Platform.

- `com.servicemesh.core.<core-version>.jar` - Base asynchronous reactor and messaging framework

- `com.servicemesh.io.<io-version>.jar` - Asynchronous I/O libraries for SSH/WinRM, HTTP, and HTTPS that include full proxy chain support (https/socks5)

- `com.servicemesh.agility.sdk.<service-version>.jar` - Services SDK base adapter/SPI and messaging definitions

- `com.servicemesh.agility.<api-version>.jar` - CSC API JAXB objects

An adapter can optionally have a dependency on another Agility Platform component such as the Agility Platform service for Azure Traffic Manager:

- `com.servicemesh.agility.distributed.sync.<sync-version>.jar` - Provides distributed mutex for synchronizing service adapter operations

A major version of the Services SDK has a dependency on a specific major/minor version of the Agility Platform REST API as the Services SDK message definitions build upon the API schema/JAXB objects.

Service Provider Asset Types

In order to support a defined service, there must be a corresponding *service provider* defined. Agility Platform incorporates the Service Provider Type asset type into its object model to support a hierarchy of defined service providers.

There are many built-in Service Provider Type subtypes, such as *Agility Platform Collector*, *Microsoft DHCP*, and *Network Service*. And the model is extended with the addition of a service adapter, where the Service Provider Type subtype that is needed to configure a service provider is defined in the Agility Platform object model, such as *AzureSQL Provider* and *AWS ELB Provider*.
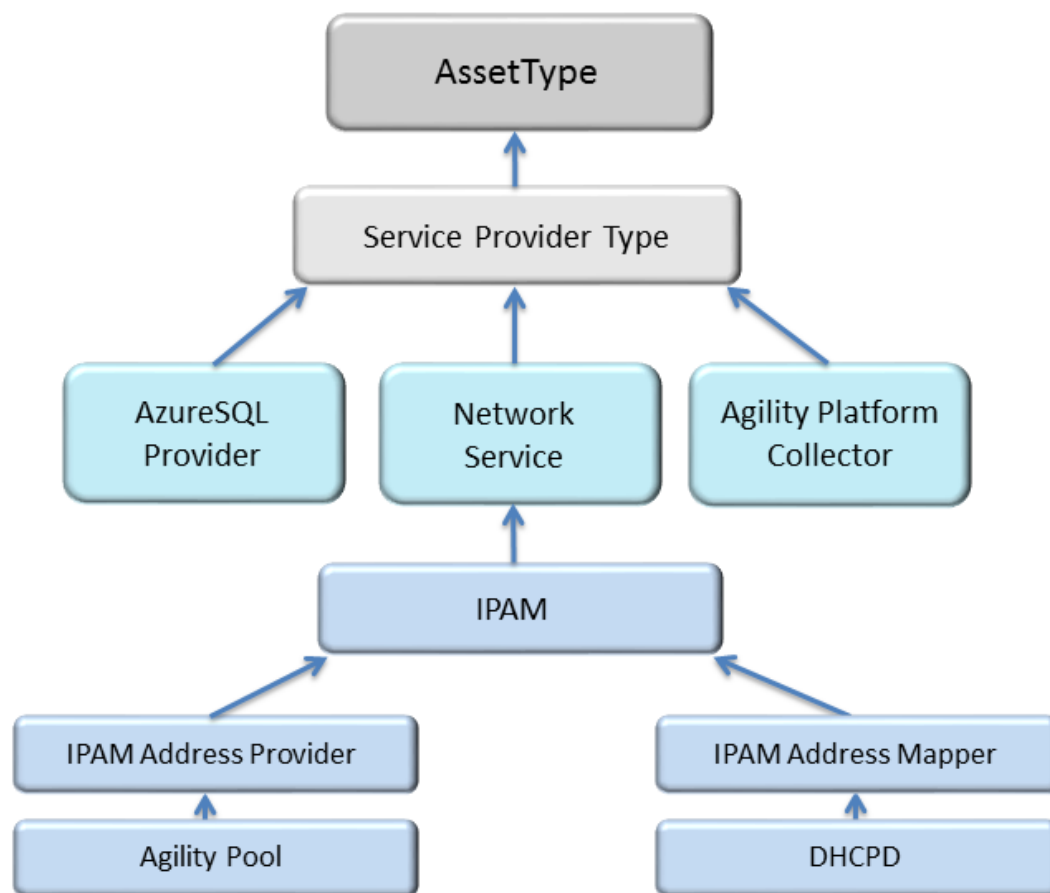
Figure 1 - Asset inheritance model for service provider types

Service providers for application services

At deployment, a service binding (*ServiceBinding*) is created to map the service require-ments/configuration (*Service*) to a specific service provider (*ServiceProvider*). Service providers can be global in scope or optionally constrained to specific cloud environ-ments/locations.

The service binding is passed to the service adapter along with the desired service config-uration (Service asset property values) on deployment to instantiate an actual service instance (such as a database account or virtual load balancer instance). This service binding can be used by the service adapter to persist the state of the service instance.

Lifecycle operations are invoked on the service adapter as instances are provisioned, started, stopped, and released to allow the adapter to inject configuration into the workload and/or make changes to the external service (such as registering the instance with a load balancer) as needed.

Service providers for network and infrastructure services

In Agility Platform, there is a class of services exists that corresponds to network and infrastruc-ture services, such IP address management (IPAM). These services provide some aspect of infrastructure automation and are typically required in certain cloud environments. An example would be IPAM via DHCP in a vSphere environment where various service providers (such as DHCPd or InfoBlox) are leveraged to manage address assignment based on organizational requirements/environment.

In this type of infrastructure/network service, the service provider is always bound to a specific cloud and set of networks on which the service is actually deployed. Instead of an application workload specifying that the service is required, the binding of the service to a deployed workload is automated based on the configuration of the service provider. Workloads deployed into an environment (network) configured for network services are bound to the service provider on deployment. Lifecycle operations are then invoked on the service adapter to perform the equivalent of the current DHCP/DNS binding functionality.

Service Asset Types

Agility Platform incorporates the Service asset type into its object model to support a hierarchy of defined application services. Functional service types, such as Data Base as a Service (DBaaS) or Load Balancer as a Service (LBaaS), cross service vendors and extend from the Service object. And a specific application service, such as Azure SQL and Amazon ELB, extends from a functional service asset type.
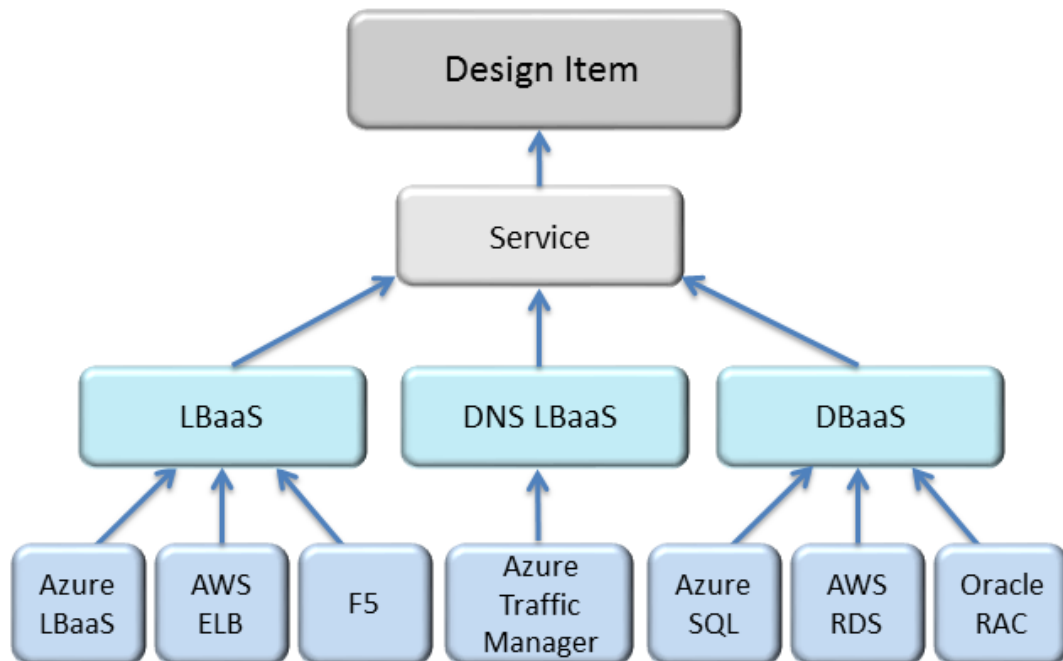
Figure 2 - Asset inheritance model for the Service asset type

Retrieving Information About Installed Adapters

An Agility Platform administrator installs a service adapter to provide the needed integration with a third-party service provider. To manage and track the developer and versions for these adapters, the Agility Platform REST API provides a method that administrators can use to return information about the installed adapters on their.Agility Platform.

> *Security:* This operation requires that the user be a member of the default *admin* user group defined in the Agility Platform.

Use the GET /adapter operation to return an *AdapterInfoList* object, such as the following example:

```
GET https://<agility_ip>:8443/agility/api/<3.2>/adapter
```

The returned *AdapterInfoList* contains an *AdapterInfo* object for each installed bundle that has "adapter" in its symbolic name, similar to the following example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AdapterInfoList xmlns="http://servicemesh.com/agility/api">
    <adapterInfo>
        <symbolicName>com.servicemesh.agility.adapters.cloud.azure-
        2.0.0</symbolicName>
        <version>3.0.0.0</version>
        <name>Agility - Azure Cloud Adapter</name>
        <vendor>Unknown</vendor>
        <category>"adapter:cloud"</category>
        <dependencies>
            <name>com.servicemesh.agility.api.script</name>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.api</name>
            <version>3.1.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.cloud.helper</name>
            <version>3.1.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.cloud.msgs</name>
            <version>3.1.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.cloud.spi</name>
            <version>3.1.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.core.async</name>
        <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.core.collections.hash</name>
```

```
                                                                                        21
        <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.collections.heap</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.collections.list</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.collections.search</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.collections.sort</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.collections.vector</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.collections</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.messaging</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.core.reactor</name>
    <version>1.0.0</version>
</dependencies>
<dependencies>
    <name>com.servicemesh.io.http</name>
</dependencies>
<dependencies>
    <name>com.servicemesh.io.proxy</name>
</dependencies>
```

```
    </adapterInfo>
    <adapterInfo>
        <symbolicName>com.servicemesh.agility.adapters.service.azure.
        trafficmanager</symbolicName>
        <version>1.0.0</version>
        <name>Agility - Azure Traffic Manger Service Provider</name>
        <vendor>CSC Agility Dev</vendor>
        <category>"adapter:service"</category>
        <description>Agility adapter to the Microsoft Azure Traffic Manager
        service. Revision r89.ae72f22.</description>
        <dependencies>
            <name>com.servicemesh.agility.adapters.core.azure.exception</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.adapters.core.azure.util</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.adapters.core.azure</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.api</name>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.distributed.sync</name>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.service.exception</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.service.msgs</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.service.operations</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
```

```
            <name>com.servicemesh.agility.sdk.service.spi</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.agility.sdk.service.util</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.core.async</name>
<version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.core.messaging</name>
            <version>1.0.0</version>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.core.reactor</name>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.io.http</name>
        </dependencies>
        <dependencies>
            <name>com.servicemesh.io.proxy</name>
        </dependencies>
    </adapterInfo>
</AdapterInfoList>
```

Each *AdapterInfo* object contains the following fields:

- **symbolicName** - Symbolic name used to uniquely identify an adapter (along with the with version)

- **version** - Version qualifier used to uniquely identify an adapter (along with the with the symbolic name)

  If the adapter has no version information, this value defaults to `0.0.0`.

- **name** - Human-readable, textual name of an adapter

- **vendor** - Human-readable, textual name that specifies the vendor of an adapter

- **category** - The category (type) of adapter, such as "adapter:service" or "adapter:cloud"

- **description** - Human-readable, textual description of an adapter.

  Not all Adapters have a description.

- ■ `dependencies` - List of package dependencies for an adapter

  This is a list of *PackageInfo* objects that have two fields (Name and Version). The version field is optional. Not all Adapters have dependencies.

Application Service Design and Deployment

The Agility Platform services architecture allows the application developer to express their requirement for externally-hosted services and their desired configuration through the application blueprint. A blueprint does not require the application developer to couple their application service to a specific implementation.

With the addition of a published service adapter, the Agility Platform has the required asset type definitions and the ability to bind an application to a vendor-specific service implement-ation. Agility Platform administrators define an application service provider (service vendor) according to the parameters for the external service. Blueprint designers can then incorporate these defined services with their application designs. At deployment, the service instance is provisioned within the topology as a runtime asset.

Blueprint Design with Application Services

In the Agility Platform, application services provide an integration path to an external service required by an application or deployment infrastructure. A defined service can be used in the composition of applications to provide support for functionality, such as a web service, database service, or message bus service.

For the application designer, the blueprint editor will expose a collection of services comprised of all sub-classes of the base service type. If the application designer requires a load balancer, they and drag an LBaaS onto the blueprint and configure the desired settings (asset type properties) without specifying the specific load balancer instance or even type that the

application should bind to. Alternately, a developer targeting EC2 may drag in an ELB load balancer and take advantage of features exposed by the ELB adapter. The following figure illustrates that a binding to an actual service instance does not occur until the blueprint is actually deployed. The deployment policy will evaluate/scope the set of available target environments based on the availability of the required service and create a binding from the generated deployment to the service provider in that cloud environment. The deployment workflow calls into the adapter to create the actual service instance for the application. Any configuration variables required by the application to connect to the service are populated by the adapter on the service binding and made available to the scripting framework.
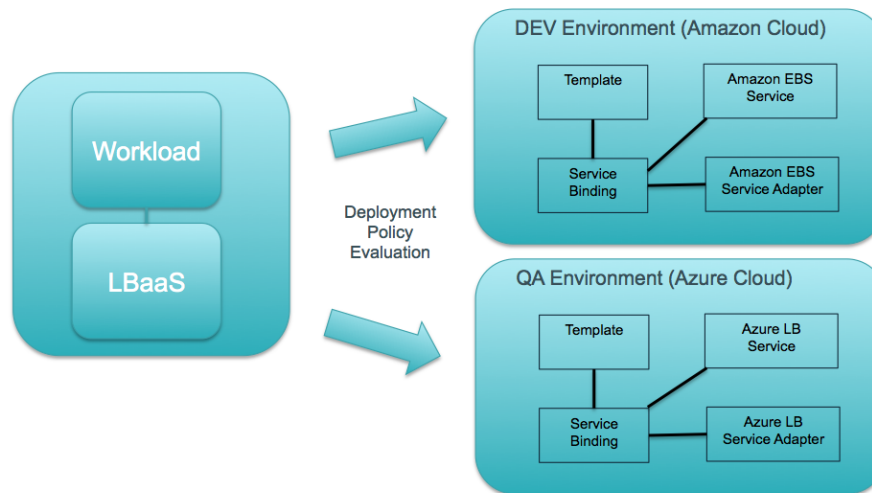
Service Lifecycle Orchestration

Upon deployment of a blueprint, the deployment policy is evaluated and a mapping established between the user-specified service requirements and a configured service provider. Agility Platform establishes this mapping as a service binding when it creates the corresponding runtime representation of the blueprint. At the time of deployment, it invokes an entry point in the service adapter to allow a service instance to be created by the adapter if required. For example, this could correspond to a virtual load balancer instance or a database account.

The following diagram illustrates the service orchestration performed by the Agility Platform.
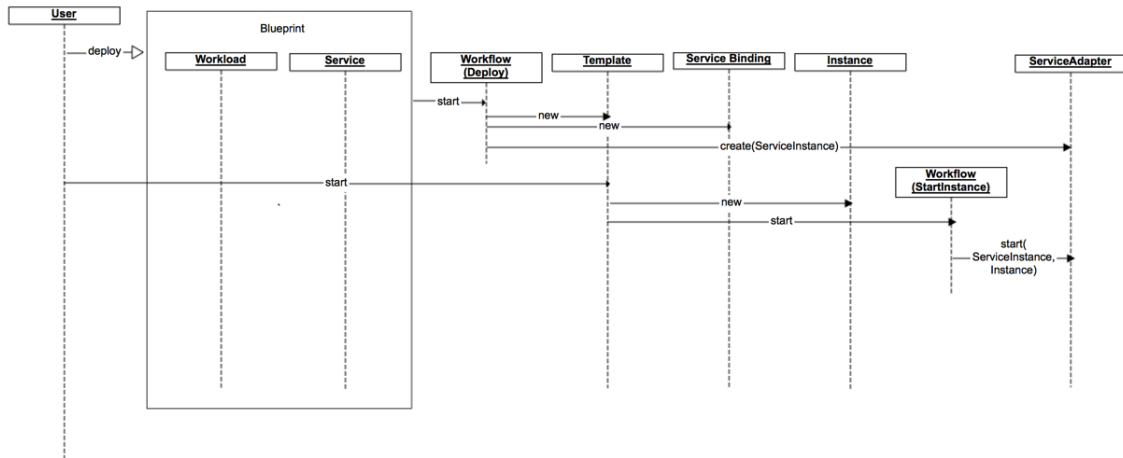
When lifecycle operations are invoked on instances corresponding to the binding, equivalent operations are invoked on the service adapter to enable the adapter to inject configuration into the instance, modify the external service instance (such as adding a VM instance to a load balancer), and any other operational requirements. These operations would be invoked from the existing VM instance-provisioning workflows as asynchronous activities that call out to the Services framework.

Network Services

Another class of services exists that correspond to the current concept of network services, such as IPAM . These services provide some aspect of infrastructure automation and are typically required in certain cloud environments. An example would be IP address assignment via DHCP in a vSphere environment. Various service providers (such as DHCPD or InfoBlox) are leveraged to manage address assignment based on organizational require-ments/environment.

However, in this case of these types of services, the service provider is always bound to a specific cloud and set of networks on which the service is actually deployed. Instead of requiring the application/workload to specify that the service is required, the binding of the service to a deployed workload is automated based on the configuration of the service provider. Workloads deployed into an environment (network) with configured network services are bound to the service provider on deployment. Agility Platform invokes lifecycle operations via the service adapter to perform the DHCP/DNS binding functionality.
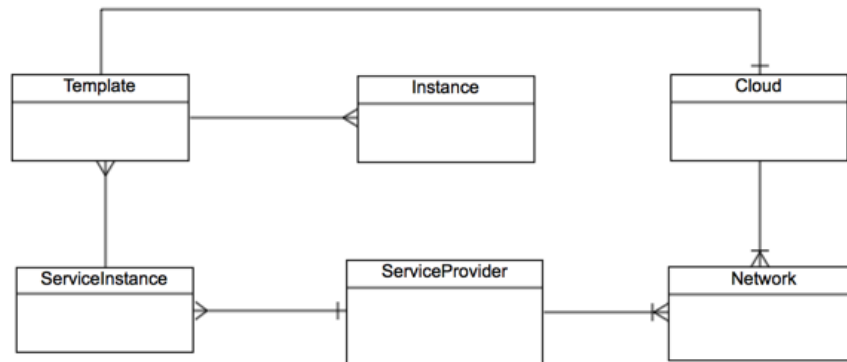
Figure 3 - Object model for the network service

Agility Platform Policy Support for Services

The following Agility Platformpolicies provide support for deployment and management of application services:

- Lifecycle (Validation and PostProcess)
- Deployment
- Firewall (limited to service adapters that support firewalls)
- Workflow

Lifecycle Policies

Lifecycle policies can be used to govern a service instance for the following service instance CRUD and compute lifecycle events:

Event	Definition
Create	The initial creation of a service instance
Update	Any update (modification) of an existing service instance
Delete	Removal of an existing asset
Provision	Creation of runtime service instance asset and service binding

Event	Definition
Release	Release of the service instance
Start	Start of the service instance
Restart	Restart of the service instance
Stop	Stop of the service instance

Lifecycle validation example

You can apply a Lifecycle Validation policy to govern service instance prior to the supported instance lifecycle events. The following example will enforce that any service instance created must start with "ServiceInstance" in the name.

```
<LifecycleCallout>
<assetType>serviceinstance</assetType>
<lifecycleEvent>Create</lifecycleEvent>
<script><![CDATA[var name = agility.params.get("this").getName();
name != null && name.startsWith("ServiceInstance");]]></script>
</LifecycleCallout>
```

Lifecycle PostProcess example

You can apply a LifecyclePostProcess policy to govern service instance after the supported instance lifecycle events. The following example will enforce that any service instance provisioned will have "ProvisionTest" added to its name.

```
<LifecycleCallout>
<assetType>serviceinstance</assetType>
<lifecycleEvent>Provision</lifecycleEvent>
<script><![CDATA[var asset = agility.params.get("this");
api=agility.getApi();
asset.setName(asset.getName() + " ProvisionTest");
api.update(asset);]]></script>
</LifecycleCallout>
```

Deployment Policies

A Deployment policy can be used to govern service instance resources, such as the service provider, during deployment. Deployment policies support the "Service" asset condition and the "Service Provider" deployment resource.

The following Deployment policy example will match the "Azure Demo" service provider to a service named "AzureDemo" and the "Azure Prod" service provider to a service named "AzureProd" upon deployment of a blueprint containing one or both of these services.

```xml
<ResourcePolicy>
    <mapping>
        <name>AzureDemo</name>
        <condition>
            <match>
                <type>application/com.servicemesh.agility.api.Service+xml</type>
                <property name="name" value="AzureDemo"/>
            </match>
        </condition>
        <inclusion>
            <item>
                <type>application/com.servicemesh.agility.api.ServiceProvider+xml
                </type>
                <name>Azure Demo</name>
                <id>4</id>
            </item>
        </inclusion>
    </mapping>
    <mapping>
        <name>AzureProd</name>
        <condition>
            <match>
                <type>application/com.servicemesh.agility.api.Service+xml</type>
                <property name="name" value="AzureProd"/>
            </match>
        </condition>
        <inclusion>
            <item>
                <type>application/com.servicemesh.agility.api.ServiceProvider+xml
                </type>
                <name>Azure Prod</name>
                <id>5</id>
            </item>
        </inclusion>
    </mapping>
</ResourcePolicy>
```

Firewall Policy

Firewall policies can be used to govern those service instances that support firewalls. The service adapter can specify this to support to use of firewall policies. If this support is provided, a Agility Platform user can assign firewall policies to the service in the blueprint editor. At runtime, the service adapter will apply the effective access list for the service instance.

Workflow Policy

Workflow policy can be used to govern the service instance lifecycle events. Agility Platform-provides the following workflow scopes by default, which can be customized according to organizational requirements:

- Service Delete
- Service Provision
- Service Release
- Service Start
- Service Stop
- Service Restart

Asynchronous Results

The Services SDK framework is based on a reactive processing model where an implementation of the reactor pattern dispatches all IO, timers, messaging, and work out of a single thread (or small pool). Additionally, calls to external service providers utilize asynchronous communication libraries (such as HTTP) that provide callback type semantics on completion. Each of these supporting libraries (such as timers, messaging, and HTTP) have their own interfaces/semantics for dealing with asynchronous completion and failures. This results in increased complexity from the perspective of the adapter writer when trying to correctly sequence asynchronous logic and propagate failure conditions.

To resolve these issues a consistent way to represent and process completion is required. The following Java Promise class provides a wrapper that can be used with any third party library or integrated into custom libraries to enable a functional approach to combining/completing asynchronous results.

```
/**
 * A promise to produce a result of type <T> at some point in the future.
 * Enables a functional approach to transforming/completing work as intermediate
 * results complete.
 *
 * @param <T>
 */
public class Promise<T>
{
/**
 * Blocks until the promise is completed and result returned or
 * failure occurs.
 */
public synchronized T get() throws Throwable;


/**
 * @return true if the request has completed.
 */
public boolean isCompleted();


/**
 * Signal that this promise is complete and notify any pending threads or mapped
results
 */
public void complete(T result);


/**
 * Set a callback to be invoked on completion of the promise.
 */
public void onComplete(final Callback<T> cb);


/**
 * @return true if the request has failed.
 */
public boolean isFailed();


/**
 * Signal that this promise is completed with an error and notify any pending
threads or mapped results
 */
```

```
public void failure(Throwable t);


/**
* Register a callback to be invoked if a failure condition occurs
*/
public void onFailure(Callback<Throwable> cb);


/**
* @return true if the request has been cancelled.
*/
public boolean isCancelled();


/**
* Signal that this promise is cancelled and notify any pending threads or pending
results
*/
public void cancel();


/**
* Set a callback to be invoked on cancellation of the promise.
*/
public void onCancel(final Callback<Void> cb);


/**
* Return a new promise that on completion of the current promise completes by
mapping the result
* using the supplied function.
*/
public <R> Promise<R> map(final Function<T,R> func);


/**
* Return a new promise that on completion of the current promise completes by
mapping the result
* using the supplied function.
*/
public <R> Promise<R> flatMap(final Function<T,Promise<R>> func);


/**
* Returns a completed promise with the specified result.
*/
```

```
public static <R> Promise<R> pure(R result);


/**

* Returns a completed promise with the specified error result.

*/

public static <R> Promise<R> pure(Throwable t);


/**

* Wrap this promise with a promise that will handle exceptions throws

*/

public Promise<T> recover(final Function<Throwable,T> func);


/**

* Returns a promise that completes when the supplied list of promises complete

*/

public static <T> Promise<List<T>> sequence(List<Promise<T>> promises);


/**

* Returns a promise that will execute work at some point in the future using the
specified reactor

*/

public static <T> Promise<T> promise(final Function0<T> func0, Reactor reactor);


/**

* Returns a promise that will execute at some point in the future using the
specified reactor

*/

public static <T> Promise<T> delayed(final Function0<T> func, long delta, Reactor
reactor);


/**

* Returns a promise that will complete at some point in the future using the
specified reactor

*/

public static <T> Promise<T> timeout(final T message, long delta, Reactor
reactor);

}
```

Third party and/or internal libraries can easily be extended or wrapped to return a `Promise<T>` rather than using a unique approach for each. As an example, the internal HTTP library is extended to include the following promise method in addition to the existing execute:

```
public interface IHttpClient

extends Closeable

{

...

public <V> Future<V> execute(final IHttpRequest request, final IHttpCallback<V>
callback);


/**

* Augment with a new method that returns a promise to the response rather than
requiring a "callback"

* object that is invoked on completion.

*/

public Promise<IHttpResponse> promise(final IHttpRequest request);

}
```

Likewise with the existing AsyncService used by the adapters for messaging:

```
public class AsyncService

{

...

public <REQ extends Request, RSP extends Response> Promise<RSP> promise(final REQ
request);

}
```

The following sections provide information about several scenarios that typically occur in an adapter and illustrate how a promise can be applied.

promise.map

In one of the simplest use cases, a service adapter often needs to map the result of an asynchronous call to an external service to an internal format. In the following example, a call is made into an adapter to test the connection to the service provider. The call should return a `Promise<ServiceProviderResponse>` that is completed when the HTTP call completes or is failed on an error condition. The `promise.map` method returns a new promise that uses the supplied function to map the result to the new type when the original promise completes.

```
@Override
    public Promise<ServiceProviderResponse> ping(final ServiceProviderPingRequest
    request)
    {
        AWSConnection connection = _factory.getConnection(request);
        Promise<IHttpResponse> promise = connection.ping();
        return promise.map(new Function<IHttpResponse,ServiceProviderResponse>()
        {
            @Override
            public ServiceProviderResponse invoke(IHttpResponse arg)
            {
                ServiceProviderResponse response = new ServiceProviderResponse();
                response.setStatus(Status.COMPLETE);
                return response;
            }
        });
    }
```

promise.sequence

Another typical use cases involves the service adapter initiating multiple asynchronous requests/calls and waiting for all of these to complete in order to accumulate a result. The `promise.sequence` method converts a list of promises to a single promise that completes with a list of results. This promise can then be combined with `promise.map` to process the results.

The following example represents an IPAM network service adapter with code removed to simplify:

```
@Override
    public Promise<InstanceResponse> preBoot(InstancePreBootRequest request)
    {
        ... // code removed

        // client wrapper around an async service provider
        AddressProvider ipam = new AddressProvider
        (addressBinding,addressProvider,service);
        List<Promise<NetworkInterface>> promises = new
        ArrayList<Promise<NetworkInterface>>();
        for(Resource resource : instance.getResources())
        {
```

```java
            if(resource instanceof NetworkInterface)
        {
            NetworkInterface nic = (NetworkInterface)resource;
            Network network = nic.getNetwork();
            Address address = nic.getAddress();
            // attempt to reserve address
            if(address != null)
            {
                Promise<NetworkInterface> promise = ipam.reserveAddress
                (instance,nic,network,address);
                promises.add(promise);
            }
            // otherwise allocate an address
            else if (network != null)
            {
                Promise<NetworkInterface> promise = ipam.allocateAddress
                (instance,nic,network);
                promises.add(promise);
            }
        }
    }


    // wait from completion of all requests
    Promise<List<NetworkInterface>> promise = Promise.sequence(promises);
    return promise.map(new Function<List<NetworkInterface>,InstanceResponse>()
    {

        @Override
        public InstanceResponse invoke(List<NetworkInterface> nics)
        {
            // update instance with updated nic(s)
            InstanceResponse response = new InstanceResponse();

            ... // code removed - do something with the nics

            response.setComplete(Status.COMPLETE);
            return response;
        }

    });
}
```

promise.flatMap

In many cases, implementing an SDK entry point requires multiple calls into the service provider. Each of the individual calls should be implemented to return a promise to the intermediate result. The results can then be combined using `flatMap`, as demonstrated in the following code example:

```
//
// Attach subnet to the load balancer
//

private Promise<InstanceResponse> enableSubnets(final InstanceRequest request,
List<String> subnetIds)
{
    try
    {
        final AWSConnection connection = _factory.getConnection(request);

... // code removed

    Promise<AttachLoadBalancerToSubnetsResponse> promise = connection.execute
    (params, AttachLoadBalancerToSubnetsResponse.class);
    return promise.flatMap(new Function<AttachLoadBalancerToSubnetsResponse,
    Promise<InstanceResponse>>()
{
    @Override
    public Promise<InstanceResponse> invoke(AttachLoadBalancerToSubnetsResponse
    lbResponse) {
        return registerInstance(request);
}
});
}
catch(Throwable t)
{
    // returns a completed promise that is failed
    return Promise.pure(t);
}
}
```

After the subnets associated with the instance are added to the load balancer, it executes the registerInstance call. `promise.flatMap` wraps the current promise with a promise that completes when the promise returned by the mapping function completes.

Error Handling

By default, all errors/exceptions are propagated up through all wrapped promises to the "top" of the promise tree/hierarchy. This behavior can be changed by installing handlers on any promise and/or wrapping any promise with a new completion that handles the error and returns the expected result type. The following method is provided for this purpose:

```
public class Promise {

    ...

    public Promise<T> recover(final Function<Throwable,T> func);

}
```

It wraps the current promise with a new promise that on a failure/exception condition invokes the supplied function to return a completion result of the expected type.

Operation Cancellation

The ability to cancel pending operations is provided as a method that will chain the cancel request down the promise tree/hierarchy to all pending operations. However, it is up to the operation to register a function that can be invoked to actually perform the cancel operations at the lowest levels.

```
public class Promise {

    ...

    public void onCancel(final Callback<Void> cb);

}
```

Installing and Updating Service Adapters

During the course of service adapter development, you can install an adapter bundle to test the adapter functionality within the Agility Platform installation that is running in your testing environment. After you install a service adapter, you can log in to the Agility Platform UI and

select the Assets item in the Admin perspective. The list of configured asset types should display the Asset Types created by the services adapter.

It is a best practice is to have the adapter code consume the resource properties file generated at build time and include versioning data in the description attribute for the asset type. After a service adapter update, you should be able to verify this information for the affected asset types.

Initial adapter installation

The initial adapter installation is very straightforward procedure—simply copy the built RPM to the Agility Platform and run the following RHEL RPM command:

```
$ sudo rpm -ivh agility-adapters-azure-trafficmanager-1.0-0.r89.ae72f22.noarch.rpm
Preparing... ######################################### [100%]
   1:agility-adapters-azure-################################################ [100%]
```

The installed package includes the versioning information from the build, similar to the following:

```
$ rpm -qa | grep trafficmanager
agility-adapters-azure-trafficmanager-1.0-0.r89.ae72f22
```

This places the adapter jar in the Agility Platform deploy directory:

```
$ ls -ltr /opt/agility-platform/deploy/*trafficman*
-rw-r--r-- 1 smadmin smadmin 439499 Apr 14 12:20 /opt/agility-
platform/deploy/com.servicemesh.agility.adapters.service.azure.trafficmanager-
1.0.0.jar
```

Simple service adapter update

Use the following RHEL RPM command to update a service adapter when there are non-major/minor version affecting changes, such as bug fixes:

```
$ sudo rpm -Uvh agility-adapters-azure-trafficmanager-1.0-0.r91.6288ee6.noarch.rpm
Preparing... ######################################### [100%]
1:agility-adapters-azure-######################################### [100%]
```

This replaces the older jar and Agility Platform immediately starts running with the updated code:

```
$ rpm -qa | grep trafficmanager
agility-adapters-azure-trafficmanager-1.0-0.r91.6288ee6
$ ls -ltr /opt/agility-platform/deploy/*trafficman*
```

```
-rw-r--r-- 1 smadmin smadmin 439509 Apr 16 12:35 /opt/agility-
platform/deploy/com.servicemesh.agility.adapters.service.azure.trafficmanager-
1.0.0.jar
```

Restarting the service framework bundle

The update command is essentially a reinstall of the bundle. A service adapter bundle interacts with the rest of the Agility Platform through the service framework bundle.

If there are issues after re-installing an adapter with significant changes, you should restart the service framework bundle using the following:

```
$ ssh -p 8022 karaf@127.0.0.1
Password:


__ __ ____
/ //_/____ _____ _/ __/
/ ,< / __ `/ ___/ __ `/ /_
/ /| |/ /_/ / / / / /_/ / __/
/_/ |_|\__,_/_/ \__,_/_/
Apache Karaf (3.0.1)
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit 'system:shutdown' to shutdown Karaf.
Hit '<ctrl-d>' or type 'logout' to disconnect shell from current session.
karaf@root()> list -s | grep framework
149 | Active | 60 | 1.0.0 | com.servicemesh.agility.adapters.service.framework
karaf@root()> restart 149
15:19:31.408 [Karaf Console ssh user karaf] INFO c.s.core.async.AsyncService -
Sent request id=1
...
```

Updating a service adapter version

If the service adapter update is a new major or minor version, do not reinstall the adapter over the existing one. Before you use the service adapter installation command, remove the old version first. The following is an example of a removal and install:

```
$ sudo rpm -e agility-adapters-azure-trafficmanager-1.0-0.r91.6288ee6
$ sudo rpm -ivh agility-adapters-azure-trafficmanager-3.2-
0.r108.bf83e21.noarch.rpm
```

Chapter 2

CSC provides the Services SDK so that you can build custom adapters for integration with any external service required by an application or the infrastructure on which it is deployed.

Using the Services SDK to develop and publish a service adapter for Agility Platform typically involves the following workflow:

1. Define the adapter components.
2. Register the adapter.
3. Synchronize the adapter.

Prerequisites

Before you begin working with the Services SDK, the Agility Platform must be installed and ready to use.

> **Note:** Refer to the *Agility Platform Installation and Setup Guide* for more information about manually installing the Agility Platform product.

Accessing the Javadoc and API Reference Information

A running instance of the Agility Platform server hosts reference documentation that provides useful information for adapter developers. Use the following URL to access links to all of the available reference information:

https://<agility_IP>:8443/doc/dev

Figure 4 - Accessing documentation used for adapter development

Use these links to access the Javadoc for each of the SDK packages, which provides summaries of classes and methods, descriptions of parameters and return values, and details about hierarchy of classes.

REST API reference

For developing automated tests for service adapter functionality, you should use v3.2 of the core Agility Platform REST API. You can access the generated API documentation for version 3.2 at the following location:

https://*<Agility_IP>*:8443/doc/rest/core/v3.2

Scroll down the list of resources on the left to access specific information about the object and its methods. Service adapter testing typically involves the following Agility Platform resources:

■ Connection

■ Instance

■ Service

■ Service Instance

■ Service Provider

■ Service Provider Type

Open Source Compliance for Service Adapters

Adapters that implement functionality using the Agility Platform Services SDK have open source license requirements that must be met before the adapter release and distribution. Because an adapter is installed and used within Agility Platform, this requirement extends beyond a standard open source license consideration. With an implementation scope that is larger than the adapter component, license violations can impact the Agility Platform itself.

Every service adapter author is required to ensure that their components are properly licensed. This includes the following:

- Selecting a license type for your adapter component. Adapters may be open source or closed (proprietary) source. If selecting open source, CSC recommends using a commonly-acceptable license, such as MIT, BSD, Apache, or Eclipse. Closed source commercial licenses are also acceptable.

- Code used to implement the service adapter must not be stolen, copied, or re-used from copyrighted or licensed open source code, except where permitted by the copyright owner. Where using publicly available code, from forums such as stackoverflow, the adapter author is solely responsible for ensuring that copyright and license requirements are met.

- All third-party libraries that are referenced/linked by the service adapter **must** be vetted for license compliance. The adapter author is solely responsible for ensuring that their adapter meets all terms required by the included or referenced libraries, such as attribution or limits on distributions.

> *Important:* Some third-party library providers are not diligent about their inclusion of libraries or dependencies. For example, although licensed as Apache 2.0, a library could have unknowingly included a dependent component that is GPL. When this occurs, the GPL component contaminates the consuming component, as well as any upstream consuming components.

Expressly prohibited licenses

The following licenses are expressly prohibited for use within Agility Platform service adapters, and from dependent libraries used by service adapters:

- GPL 2.0 - GNU Public License
- GPL 3.0
- AGPL - Afferro GPL

- LGPL - Limited GPL

Acceptable licenses

The following licenses are expressly allowed and preferred for use within Agility Platform service adapters, and from dependent libraries used by service adapters:

- MIT
- BSD
- Apache 2.0
- Eclipse Foundation License

Other licenses

The service adapter author should vet all other licenses using knowledgeable advisors. CSC suggests the following advisors:

- CSC Open Source Program
- Agility Platform security team
- Source Auditor - http://www.sourceauditor.com/

Service Provider Credential Security

As a service adapter developer, you will define new subtype asset types for a service, which typically inherit from the *service*, *serviceprovidertype*, and/or *designconnection* base asset types. A new asset type can have a property that represents sensitive data, such as a password. A property that can impact the operational security of an enterprise should have its value hidden for direct display and should use encryption for storage.

When you specify a new asset types in a `RegistrationRequest` and a sensitive property requires this security, make its type encrypted. Agility Platform handles a property with the encrypted type by displaying asterisks for its value and encrypting the value within the Agility Platform database. The following is an example:

```
Link encryptedType = new Link();
encryptedType.setName("encrypted"); encryptedType.setType("application/" +
PropertyType.class.getName() + "+xml");
// Define a required access code property for my service provider
PropertyDefinition accessCodePD = new PropertyDefinition();
accessCodePD.setName("access-code"); accessCodePD.setDescription("Service Provider
Access Code.");
```

```
accessCodePD.setDisplayName("Access Code");

accessCodePD.setReadable(true);

accessCodePD.setWritable(true);

accessCodePD.setMaxAllowed(1);

accessCodePD.setMinRequired(1);

accessCodePD.setPropertyType(encryptedType);

AssetType myProvider = new AssetType();

...

myProvider.getPropertyDefinitions().add(accessCodePD);
```

The Agility Platform service adapter framework decrypts the value when passing it in a `ServiceProviderRequest` object to the service adapter.

Service Adapter Registration

The `ServiceAdapter` base class registers the adapter as an OSGi service and uses the OSGi service registry to look up an `AsyncService` implementation provided by the corresponding version of the Services SDK.

The adapter must include an implementation of the `ServiceAdapter.getRegistrationRequest()` function to populate the `RegistrationRequest` message with the relevant settings that define the adapter capabilities. After it receives the registration request, the Services framework queries the database and creates new or updates current entries required to register the service type and associated asset types and capabilities.

To construct and register a service adapter:

1. Define the service provider type asset type(s).
2. Define the service asset type(s).
3. Define allowed connections/dependencies.
4. *(Optional)* Define custom actions.

Guidelines for Service Adapter Development

To ensure consistency and compatibility across the adapters used within a single Agility Platform installation but developed by different groups, there are guidelines for defining these adapters and their defined asset types. When you define the service properties for a service adapter, these entities are displayed in the Agility Platform user interface, stored in the Agility Platform database, and presented through API calls. Using these guidelines when developing an adapter provides a more consistent administrator and user experience and helps to maintain supportability for a published adapter.

Adapter Naming Conventions

Use the following guidelines to define the adapter within the Adapter MANIFEST.MF file:

- *Bundle-Version*: Specify the adapter version, with an initial adapter version of "1.0.0"
- *Bundle-Vendor*: Designate the organization that developed the adapter.

  The development group should determine the appropriate designation to identify their development group and use it consistently. This will fall into one of three categories:

  ♦ Adapters developed by the core Agility Platform or Agility Platform adapters teams, which is designated by `CSC Agility Dev`.

  ♦ Adapters developed by other groups within CSC, which should be designated appropriately (such as `EBG Pro Serve` for a Professional Services teams)

  ♦ Adapters developed by third parties outside of CSC, which should be designated by organization name and business line, if applicable (such as `EMC VIPR`, `F5 BigIP`, or `Frysoft`)

  > *Note:* The vendor information for the adapter will be used to determine any Support agreements that are in place, including those specified by contractual relationships with partners or through paid, custom Professional Services projects.

- *Bundle-Category*: Use one of the following as an identifier for the adapter type:

  ♦ `"adapter:service"` identifies a bundle containing an adapter to support a service

  ♦ `"adapter:cloud"` for a bundle containing an adapter to support a cloud provider

  ♦ `"adapter:core"` for a bundle containing core functionality that can be utilized by a service or cloud adapter

- *Bundle-Description*: Include build information within the text, which makes it available via the `GET /adapters` REST call.

For more information about using this REST API operation and the return object, see "Retrieving Information About Installed Adapters" on page 19.

Asset Naming Conventions

Use the following guidelines to define the service asset types that are created in Agility Platform when the service adapter is installed:

■ *Service Name*: Include the version number in the service name in order to differentiate between service adapters.

If there are two versions of the same adapter installed, users can identify the service that they need by its version number.

■ *Service Type*: The name (not the display name), which is used in the blueprint editor.

Agility Platform users can search for these items, so this should be spelled out (such as "my-cool-service").

■ *Property Names*: The name of a service property, which is used to define the service.

Hyphen-separated (such as "some-cool-property") is preferred over camel case.

■ *Asset Type Descriptions*: The descriptive text for the asset type.

It is a best practice to include the bundle vendor and version. Including this information makes it readily available in the Agility Platform user interface for administrators working with asset types.

This information can be specified dynamically (see the following section).

Copyright and Licensing for Adapter Files

Adapter development teams within CSC should include a LICENSE file at the root of the project and add the following copyright message to the source files:

```
Copyright (c) 2013-Present Computer Sciences Corporation
```

When you include the LICENSE text file in the project, the contents of this file should include the following:

```
<Project Name> -
    Copyright (C) 2008-2013 ServiceMesh, Inc
    Copyright (C) 2013-Present Computer Sciences Corporation
Computer Sciences Corporation can be contacted at: info@csc.com


Licensed under the Apache License, Version 2.0 (the "License"); you may not
```

```
use or distribute <Project Name> except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

If the adapter is part of an Open Source project, the following text is also included in the `LICENSE` text file:

```
To contribute to this project, please see Contributor License Agreement (CLA) and
other open source information at:
https://github.com/csc/csc-open-source
```

The ServiceAdapter Base Class

The `ServiceAdapter` abstract base class contains abstract methods for the core messaging framework and OSGi service registration. The `ServiceAdapter` class implements the OSGi `BundleActivator` interface, which handles the adapter registration with the core platform and the registration of the message handlers for supported message types.

Service adapters should extend the following service adapter class and implement the abstract methods.

```
public abstract class ServiceAdapter implements BundleActivator
{
/**
* @return Should return the name associated with the service provider type. Used
to register the osgi service.
*/
public abstract String getServiceProviderType();


/**
* @return A descriptive name for the adapter.
*/
public abstract String getAdapterName();
/**
```

```
* Required meta-data to describe capabilities of the adapter/service provider. The
returned RegistrationRequest
* is sent to the agility platform and the onRegistration method called with the
response.
*
* @return
*/
public abstract RegistrationRequest getRegistrationRequest();


/**
* Provides any information requested by the adapter during the registration
request including
* the persistent identifier for the service provider.
*
* @param response
*/
public abstract void onRegistration(RegistrationResponse response);


/**
* @return
* An optional interface that implements hooks in instance lifecycle for workloads
* dependent on the service.
*/
public IInstanceLifecycle getInstanceOperations() { return null; }


/**
* @return
* A required interface used to manage the service provider.
*/
public abstract IServiceProvider getServiceProviderOperations();


/**
* @return
* A required interface to manage the lifecycle of bindings (service instances) to
the
* service provider.
*/
public abstract IServiceInstance getServiceInstanceOperations();


/**
```

```
* @return

* An optional interface to receive lifecycle (CRUD) notifications for specific

* asset types. An adapter will send a list of asset types it's interested in

* receiving CRUD event messages for when it builds it's initial registration
message.

*/

public IAssetLifecyle getAssetNotificationOperations() { return new
AssetOperations();}


}
```

The RegistrationRequest Class and Object

The `com.servicemesh.agility.sdk.service.msgs.RegistrationRequest` class is the container for a service adapter to specify its core data. Every service adapter must override `com.servicemesh.agility.sdk.service.spi.ServiceAdapter.getRegistrationRequest()` and return a populated `RegistrationRequest` object. The `getRegistrationRequest()` method is invoked during the adapter bundle activation.

The `RegistrationRequest` object includes the following attributes:

Attribute	Description	Required?
String name	Adapter name	Yes
String version	Adapter version	Yes
List <ServiceProviderType> serviceProviderTypes	Definition of the service provider type(s) exposed by the adapter	No
List<AssetType> assetTypes	Asset type definitions for service type(s)	Yes
List<Property> properties	List of Property objects that describe the adapter *Note:* This is for self-documenting an adapter and the Property objects are not utilized at runtime.	Yes
List<Property> settings	List of Property objects for configuring the adapter *Note:* The configuration Property objects are included in the com.servicemesh.agility.sdk.service.msgs. ServiceProviderRequest passed into an adapter at runtime.	Yes

Attribute	Description	Required?
List<String> supportedInterfaces	List of Agility interfaces for which the adapter is to be notified when a CRUD operation occurs	Yes
List<String> assetLifecycles	List of Agility asset classes for which the adapter is to be notified when a CRUD operation occurs	Yes

Most service adapters will need to define the optional `serviceProviderTypes` and `assetTypes` attributes. For detailed examples of these, see "The ServiceProviderType Asset Type" on page 55 and "Registration of the Service Asset Type" on page 65

In addition to any specified interfaces, the `supportedInterfaces` attribute is automatically populated based on the operations implemented by an adapter:

Implemented operations	Interface
Interface	servicemesh.agility.sdk.service.spi.IInstanceLifecycle
ServiceProvider	servicemesh.agility.sdk.service.spi.IServiceProvider
ServiceInstance	servicemesh.agility.sdk.service.spi..IServiceInstance

The `assetLifeCycles` attribute allows the adapter to be notified when specific types of Agility Platform assets change, such as the following example:

```
registration.getAssetLifecycles().add("com.servicemesh.agility.api.Network");
```

Including Dynamic Service Adapter Information

The following example uses Ant as the build tool and a service adapter written in Java. The application is configured so that the version information is changed in one place, the Build.xml file. There are three files that are configured to support this: Build.xml, MANIFEST.MF, and MyAdapter.java.

Build.xml

The following example provides excerpts from a typical Ant Build.xml file. Specifically, it replaces `%bundle_version` with the value of bundle.version as the combination of the major, minor, build version numbers. This way, the version number is specified in one place (Build.xml) and developers do not have to remember to update the version in a file when it is time to build a new version of the adapter.

```
<!-- version properties -->
    <property name="version.major" value="1" />
    <property name="version.minor" value="0" />
    <property name="version.build" value="0" />
    <property name="bundle.version"
    value="${version.major}.${version.minor}.${version.build}" />


...


<target name="deploy" depends="compile,deploy_unit_test,git-info"
description="Generate and deploy bundle">
    <delete file="${bin}/${bundle.jar}" />
    <property name="manifest" value="${bin}/META-INF/MANIFEST.MF"/>
    <copy file="META-INF/MANIFEST.MF" tofile="${manifest}" overwrite="true"/>
    <replace file="${manifest}" token="%bundle_version"
    value="${bundle.version}"/>
    <echo file="${manifest}" append="true" message="Bundle-Description: Agility
    adapter for the MyAdapter service. Revision r${git.summary}.${line.separator}
    "/>
    <property name="propfile" value="${bin}/resources/MyAdapter.properties"/>
    <echo file="${propfile}">adapter.vendor=CSC Agility Dev
    adapter.version=${bundle.version}
    adapter.revision=r${git.summary}
    </echo>
    <jar destfile="${bin}/${bundle.jar}" manifest="${manifest}" basedir="${bin}">
        <fileset dir="${bin}">
```

```
            <include name="**/*.class"/>
            <include name="**/*.properties"/>
        </fileset>
        <fileset dir="${lib}" includes="*.jar"/>
    </jar>
</target>
```

MANIFEST.MF

This excerpt demonstrates appending the Bundle-Description to the Manifest file and includes the current git revision in the description. The MANIFEST.MF file has no entry for `Bundle-Description`.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyAdapter
Bundle-SymbolicName: com.company.product.adapters.MyAdapter
Bundle-Version: %bundle_version
Bundle-Activator: com.company.product.adapters.MyAdapter
Bundle-Vendor: CSC Agility Dev
```

MyAdapter.java

The Build.xml creates a properties file called `MyAdapter.properties`, which contains the vender, version, git revision information. This excerpt of MyAdapter.java saves these property values from the properties file into Java variables. It creates the version information string that is inserted into the Agility Platform AssetType descriptions. The version information string uses a "`(version gitRevision vendor)`" format. It also appends the version information string to the AssetType name and saves that as the description to be used for the AssetType.

```
public static final String SERVICE_PROVIDER_NAME = "My Service Provider";
    public static final String SERVICE_PROVIDER_DESCRIPTION;
    public static final String SERVICE_PROVIDER_TYPE = "my-service-provider";
    public static final String SERVICE_PROVIDER_VERSION;

    public static final String SERVICE_TYPE = "my-service";
    public static final String SERVICE_NAME = "My Service";
    public static final String SERVICE_DESCRIPTION;

    static {
        String PROP_FILE = "/resources/MyAdapter.properties";
        Properties props = new Properties();
```

```
        try {
            InputStream rs =
             AzureSQLAdapter.class.getResourceAsStream(PROP_FILE);
            if (rs != null)
                props.load(rs);
            else
                logger.error("Resource not found " + PROP_FILE);
        }
        catch (Exception ex) {
            logger.error("Failed to load " + PROP_FILE + ": " + ex);
        }
        SERVICE_PROVIDER_VERSION = props.getProperty("adapter.version", "0.0.0");
        String revision = props.getProperty("adapter.revision", "");
        String vendor = props.getProperty("adapter.vendor", "");

        StringBuilder sb = new StringBuilder();
        sb.append(" (").append(SERVICE_PROVIDER_VERSION);
        if (! revision.isEmpty()) {
            sb.append(" ").append(revision);
        }
        if (! vendor.isEmpty()) {
            sb.append(" ").append(vendor);
        }
        sb.append(")");
        SERVICE_PROVIDER_DESCRIPTION = SERVICE_PROVIDER_NAME + sb.toString();
        SERVICE_DESCRIPTION = SERVICE_NAME + sb.toString();
    }
```

The ServiceProviderType Asset Type

The `ServiceProviderType` specifies the communications mechanism for a service adapter to manage a service in a cloud. The `RegistrationRequest assetTypes` attribute should be populated with an Asset Type that represents the communications parameters for a service provider. This asset type extends from the base ServiceProviderType asset type. For more information, see "Service Provider Asset Types " on page 17.

A Service Provider Type defined by a service adapter becomes available in the Agility Platform service provider type administration dialog upon activation of the adapter bundle.

For the following example, assume that a service provider uses a certificate with a private key to communicate with a cloud that provides the service:

```
String X_PROPERTY_TYPE = "application/" + PropertyType.class.getName() + "+xml";
PropertyDefinition certificatePD = new PropertyDefinition();
certificatePD.setName("certificate");
certificatePD.setDescription("Certificate for service provider to access cloud.");
certificatePD.setDisplayName("Certificate");
certificatePD.setReadable(true);
certificatePD.setWritable(true);
certificatePD.setMaxAllowed(1);
Link binaryType = new Link();
binaryType.setName("binary");
binaryType.setType(X_PROPERTY_TYPE);

certificatePD.setPropertyType(binaryType);
PropertyDefinition privateKeyPD = new PropertyDefinition();
privateKeyPD.setName("private-key");
privateKeyPD.setDescription("Private key for service provider certificate.");
privateKeyPD.setDisplayName("Private Key");
privateKeyPD.setReadable(true);
privateKeyPD.setWritable(true);
privateKeyPD.setMaxAllowed(1);
Link encryptedType = new Link();
encryptedType.setName("encrypted");
encryptedType.setType(X_PROPERTY_TYPE);
privateKeyPD.setPropertyType(encryptedType);
```

```
AssetType svcProviderAT = new AssetType();

svcProviderAT.setName("my-service-provider");

svcProviderAT.setDisplayName("Example Service Provider");

svcProviderAT.setDescription("Service provider definition example");

svcProviderAT.getPropertyDefinitions().add(certificatePD);

svcProviderAT.getPropertyDefinitions().add(privateKeyPD);


// serviceprovidertype is the base type for the service provider type
// created by a service adapter
Link parent = new Link();

parent.setName("serviceprovidertype");

parent.setType("application/" + ServiceProviderType.class.getName() + "+xml");

svcProviderAT.setSuperType(parent);


registration.getAssetTypes().add(svcProviderAT);
```

A corresponding `ServiceProviderType` object should also be added to the
`RegistrationRequest serviceProviderTypes` attribute, such as the following example:

```
ServiceProviderType serviceProviderType = new ServiceProviderType();

serviceProviderType.setName(svcProviderAT.getDisplayName());

serviceProviderType.setDescription(svcProviderAT.getDescription());


// This service provider must be directly associated with the service
// provider type AssetType included in the RegistrationRequest.
Link assetType = new Link();

assetType.setName(svcProviderAT.getName());

assetType.setType("application/" + AssetType.class.getName() + "+xml");

serviceProviderType.setAssetType(assetType);


// This service provider must be associated with a service. The name should
// also be the name of the AssetType added to the RegistrationRequest for
// the service.
Link serviceType = new Link();

serviceType.setName("my-service");

serviceProviderType.getServiceTypes().add(serviceType);


// This example service provider only relies on a certificate with a
// private key. Disable the four default properties for a service provider
// so that the Agility UI does not prompt for them.
```

```
serviceProviderType.getOptions().add(ServiceProviderOption.NO_HOSTNAME);

serviceProviderType.getOptions().add(ServiceProviderOption.NO_NETWORKS);

serviceProviderType.getOptions().add(ServiceProviderOption.NO_USERNAME);

serviceProviderType.getOptions().add(ServiceProviderOption.NO_PASSWORD);


registration.getServiceProviderTypes().add(serviceProviderType);
```

The Base Service Asset Type Inheritance Model

In the Agility Platform object model, an asset type can inherit from another type. This base type is the *supertype* for the inheriting asset type. The inheriting asset type is the *subtype* for the supertype. The subtype inherits properties from its supertype, in addition to those properties that are define explicitly for the subtype.

> *Important:* When you define a new service asset type as a subtype, do not create a property for your service that has the same name as one of the properties that it inherits.

The Service framework installed within the Agility Platform defines an existing inheritance model that extends for the Service asset type so that service adapter developers can easily extend from common functional service asset types to define vendor-specific service asset types.
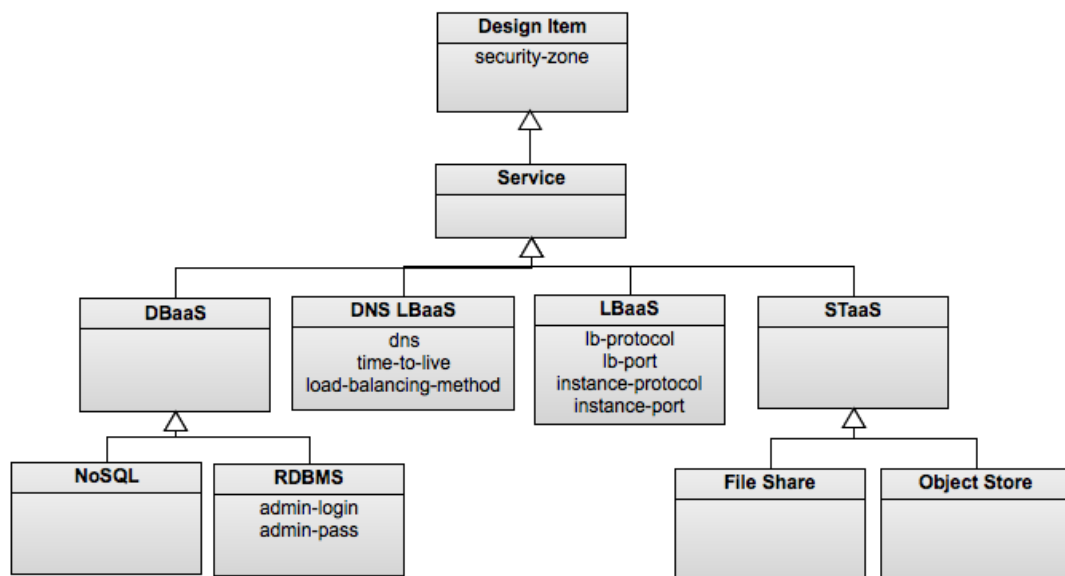
Figure 5 - Defined base service asset types

Design Item Properties

`DesignItem` is the base type of all asset types that can be used in the blueprint editor.

Inherited properties: None

Defined properties:

security-zone

Name	security-zone
Display Name	Security Zone
Description	The security zone associated with the asset
Min Required	0
Max Allowed	1
Property Type	Security Zone
Constraints	Possible Values: Management, Restricted, Secured, Internal Controlled, Controlled, External Controlled, External Uncontrolled

Service Inheritance and Properties

`Service` is the base asset type for all Services. Any asset type that inherits from Service is displayed in the Services list in the blueprint editor.

Inherited properties: security-zone

Defined properties: none

Type inheritance: Design Item >> Service

DBaaS Inheritance and Properties

`DBaaS` is the base asset type for all Database-as-a-Service asset types.

Inherited properties: security-zone

Defined properties: none

Type inheritance: Design Item >> Service >> DBaaS

NoSQL Inheritance and Properties

`NoSQL` is the base asset type for all NoSQL Database as a Service service types.

Inherited properties: security-zone

Defined properties: none

Type inheritance: Design Item >> Service >> DBaaS >> NoSQL

RDBMS Inheritance and Properties

`RDBMS` is the base type of Relational-Database-Management-as-a-Service asset types.

Type inheritance: Design Item >> Service >> DBaaS >> RDBMS

Inherited properties: security-zone

Defined properties:

admin-login

Name	admin-login
Display Name	Admin Login
Description	Specifies the username of the Admin User.
Min Required	1
Max Allowed	1
Property Type	String

admin-pass

Name	admin-pass
Display Name	Admin Password
Description	Specifies the password of the Admin User.
Min Required	1

Max Allowed	1
Property Type	Encrypted

DNS LBaaS Inheritance and Properties

`dns-lbaas` is the base type for all DNS Load Balancer-as-a-Service asset types.

Type inheritance: Design Item >> Service >> DNS LBaaS

Inherited properties: security-zone

Defined properties:

dns

Name	dns
Display Name	Domain Name
Description	Specifies the name of the domain.
Min Required	1
Max Allowed	1
Property Type	String
Constraints	Maximum length:253

time-to-live

Name	time-to-live
Display Name	DNS Time-to-Live (Seconds)
Description	DNS TTL value, in seconds, to cache DNS entries
Min Required	0
Max Allowed	1

Property Type	Integer

Constraints	Maximum digits: 20
	Range: 30-999999.
	Default value: 30

load-balancing-method

Name	load-balancing-method

Display Name	Load Balancing Method

Description	Specifies the load balancing method to use to distribute connections

Min Required	1

Max Allowed	1

Property Type	String

Constraints	Possible Values: Performance, Failover, RoundRobin

LBaaS Inheritance and Properties

`lbaas` is the base type for all non-DNS Load Balancer-as-a-Service asset types.

Type inheritance: Design Item >> Service >> LBaaS

Inherited properties: security-zone

Defined properties:

lb-protocol

Name	lb-protocol

Display Name	LB Protocol

Description	Front-end protocol exposed by the load balancer

Min Required	0

Max Allowed	16
Property Type	Integer
Constraints	Range: 1-65534 Max Digits: 20

lb-port

Name	lb-port
Display Name	LB Port
Description	Front-end port exposed by the load balancer
Min Required	0
Max Allowed	16
Property Type	Inetger
Constraints	Range: 1-65534 Max Digits: 20

instance-protocol

Name	lb-protocol
Display Name	Instance Protocol
Description	Backend protocol used to communicate to the guest instance
Min Required	0
Max Allowed	16
Property Type	String
Constraints	Possible Values: HTTP, HTTPS, TCP, SSL

instance-port

Name	instance-port
Display Name	Instance Port
Description	Backend port used to communicate to the guest instance
Min Required	0
Max Allowed	16
Property Type	Inetger
Constraints	Range: 1-65534 Max Digits: 20

STaaS Properties

StaaS is the base asset type for all Storage-as-a-Service service asset types.

Inherited properties: security-zone

Defined properties: none

Type inheritance: Design Item >> Service >> STaaS

File Share Inheritance and Properties

fileshare is the base asset type for all file share service asset types.

Inherited properties: security-zone

Defined properties: none

Type inheritance: Design Item >> Service >> STaaS >> File Share

Object Store Inheritance and Properties

objectstore is the base asset type for all object store service asset types.

Inherited properties: security-zone

Defined properties: none

Type inheritance: Design Item >> Service >> STaaS >> Object Store

Extending a Base Service Asset Type

It is relatively simple to create a new Service asset type that is derived from a general service type, such as the existing base types (see "The Base Service Asset Type Inheritance Model" on page 57). The primary task is setting the supertype of your service to the desired parent asset type. For example, if you want to create a Acme-Database-Adapter with RDBMS as its parent, you would set the supertype of the Acme-Database-Adapter to RDBMS.

Use the `getRegistrationRequest` method of the main adapter class to set the supertype, similar to the following example:

```
@Override
    public RegistrationRequest getRegistrationRequest()
    {
        RegistrationRequest registration = new RegistrationRequest();

        Link parentServiceType = new Link();
        parentServiceType.setName("rdbms");
        parentServiceType.setType("application/" + Service.class.getName() +
        "+xml");

        AssetType acmeDatabase = new AssetType();
        acmeDatabase.setName("acme-database-adapter");
        acmeDatabase.setDisplayName("Acme Database Adapter");
        acmeDatabase.setSuperType(parentServiceType);
        registration.getAssetTypes().add(acmeDatabase);
        return registration;

    }
```

For more information about using `getRegistrationRequest`, see "Registration of the Service Asset Type" on page 65.

Registration of the Service Asset Type

The Service Asset Type represents the cloud service supported by a service adapter. The `RegistrationRequest assetTypes` attribute should be populated with an Asset Type that includes the properties of a service. This asset type extends from the base Service asset type. For more information, see "Extending a Base Service Asset Type" on page 64.

A Service defined by a service adapter becomes available in the Agility Platform blueprint editor upon activation of the adapter bundle. After adding the service to a blueprint, users can specify property values for the service.

For the following example, assume that the service uses a single string property:

```
PropertyDefinition svcStringPD = new PropertyDefinition();
svcStringPD.setName("my-service-string-property");
svcStringPD.setDisplayName("Example Service String Property");
svcStringPD.setDescription("String property definition example");
svcStringPD.setReadable(true);
svcStringPD.setWritable(true);
svcStringPD.setMaxAllowed(1);
Link string_type = new Link();
stringType.setName("string-any");
stringType.setType(X_PROPERTY_TYPE);
svcStringPD.setPropertyType(stringType);

AssetType serviceAT = new AssetType();
serviceAT.setName("my-service");
serviceAT.setDisplayName("Example Service");
serviceAT.setDescription("Service definition example");
serviceAT.getPropertyDefinitions().add(svcStringPD);

// service is the base type for this example service. Any of the Agility
// pre-defined asset types that inherit from service could be used here.
Link svcParent = new Link();
svcParent.setName("service");
svcParent.setType("application/" + Service.class.getName() + "+xml");
serviceAT.setSuperType(svcParent);
```

```
registration.getAssetTypes().add(serviceAT);
```

Registration for Additional Blueprint Editor Support

In addition to asset type properties, the Blueprint Editor in the Agility Platform supports the definition of Firewalls, Policies, and Variables for a service. To enable these additional configuration types for a service within a blueprint design, you must add the editor type to the Editors list in the AssetType that is returned in the `RegistrationRequest` by `getRegistrationRequest`. For more information about this method, see "The RegistrationRequest Class and Object" on page 50.

The following code example enables the editors for Firewalls, Policies, and Variables for a service:

```
import com.servicemesh.agility.api.Editor;


@Override
public RegistrationRequest getRegistrationRequest()
{
    RegistrationRequest registration = new RegistrationRequest();
    AssetType serviceType = new AssetType();
    // setup service properties...
    serviceType.getEditors().add(Editor.FIREWALL);
    serviceType.getEditors().add(Editor.POLICY);
    serviceType.getEditors().add(Editor.VARIABLES);
    registration.getAssetTypes().add(serviceType);
    return registration;

}
```

Design Connection Asset Type

Agility Platform incorporates the Design Connection asset type into its object model to support a hierarchy of defined dependencies between blueprint design elements. A *design connection* indicates the systems on which a machine relies for script variable values or services. A dependent node cannot start until all dependencies have successfully started and executed all startup scripts. Additionally, if a dependency stops or restarts, all dependent nodes execute all reconfigure scripts included in assigned packages.

A subtype of Design Connection, such as Azure Traffic Manager Connection or Amazon ELB Connection, is used by Agility Platform administrators to define a valid connection between the service and other blueprint design objects.
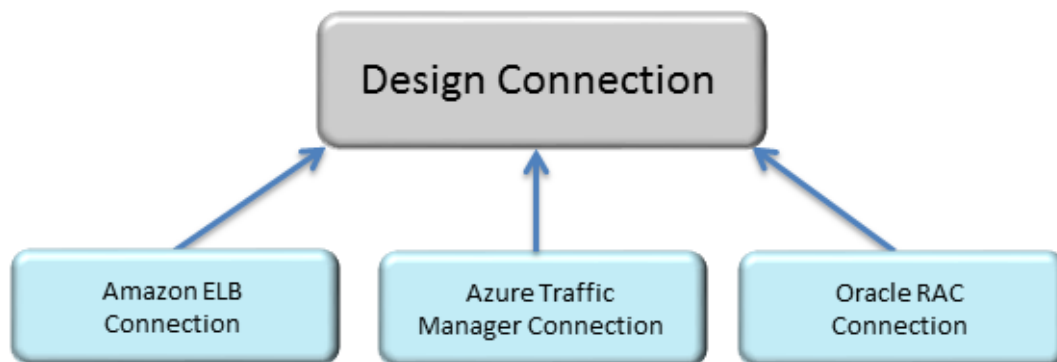
Figure 6 - Asset inheritance model for the Design Connection asset type

Supporting Connections for a Service

Blueprint designers define relationships between blueprint design elements using connections. A connection indicates the systems on which a VM relies for script variable values or services. A dependent node cannot start until all dependencies have successfully executed all startup scripts. Additionally, if a dependency stops or restarts, all dependent nodes execute all reconfigure scripts included in assigned packages.

By default, the Agility Platform installs with two basic design connections defined:

- Workload -> Workload: a workload can depend on another workload
- Workload -> Blueprint: a workload can depend on an embedded blueprint

A service adapter should define the valid connections between an application service and the workloads, child blueprints, and other application services that are supported within a blueprint design. For a subtype of the Design Item supertype, you can define the valid source/destination connections that will be supported in the Blueprint editor.

To support a connection to or from the service, you create a `ConnectionDefiniton`. For all connections, the Source asset depends on the Destination asset. In the Agility Platform user interface, the destination is the asset with the head of the arrow.

To define a connection, will must populate the following five fields at a minimum:

Field	Usage
Name	The internal name of the connection It is recommended that you use a hyphen delimited name where the name is the type of the source DesignItem followed by the type of the destination DesignItem. For example, `workload-service` would be a connection from a workload to a service.
Display Name	The name of the connection that is displayed in the Agility Platform user interface (Blueprint Editor)
Source Type	A Link to the type of the source DesignItem
Destination Type	A Link to the type of the destination DesignItem
Connection Type	A Link to the type of the connection You should use the DesignConnection type if you do not need a custom connection type.

Defining a Service Connection

Use the `getRegistrationRequest` method to create the connection definition and add it to the Service asset type. If the service is the source in the connection, add the connection definition to the Service AssetType `SrcConnections` list. If the service is the destination in the connection, add the connection definition to the Service AssetType `DestConnections` list. For more information about using this method, see "Extending a Base Service Asset Type" on page 64.

The following example defines a Service Connection where the service that can depend on workloads and also allows workloads to depend on the service.

```
Link workload_link = new Link();
workload_link.setName("designworkload");
workload_link.setType("application/" + Workload.class.getName() + "+xml");
```

```
Link myService_link = new Link();
myService_link.setName(SERVICE_TYPE);
myService_link.setType("application/" + Service.class.getName() + "+xml");


Link connection_link = new Link();
connection_link.setName("designconnection");
connection_link.setType("application/" + Service.class.getName() + "+xml");


ConnectionDefinition workload_to_myservice = new ConnectionDefinition();
workload_to_myservice.setName("workload-myservice");
workload_to_myservice.setDisplayName("Dependency on My Service");
workload_to_myservice.setSourceType(workload_link);
workload_to_myservice.setDestinationType(myService_link);
workload_to_myservice.setConnectionType(connection_link);


ConnnectionDefinition myservice_to_workload = new ConnectionDefiniton();
myservice_to_workload.setName("myservice-workload");
myservice_to_workload.setDisplayName("Dependency on Workload");
myservice_to_workload.setSourceType(myService_link);
myservice_to_workload.setDestinationType(workload_link);
myservice_to_workload.setConnectionType(connection_link);


AssetType myService = new AssetType();
myService.getDestConnections().add(workload_to_myservice);
myService.getSrcConnections().add(myservice_to_workload);
```

Using a Custom Design Connection Type

When you are providing support for connections to or from a service, you can use the base `DesignConnection` type or define a custom connection type. All custom connection types must have `DesignConnection` as the top-level parent. It possible to have a custom connection type inherit from another custom connection type. For example, you could have an inheritance chain similar to DesignConnection -> CustomA -> CustomB.

The main reason for using a custom connection type is to support properties associated with the connection. For example you might want a property named "Weight" so the system can determine which connections are more important.

Use the `getRegistrationRequest` method to define the custom connection type and add it to the AssetTypes list of the `RegistrationRequest`. For more information about using this method, see "Extending a Base Service Asset Type" on page 64.

> **!** ***Important:*** If you are defining multiple AssetTypes and want to allow custom connections between them, you must provide those AssetTypes to the `RegistrationRequest` twice. The first time without the connections and a second time with the connections. This is because the `ConnectionDefinition` expects both of the AssetTypes to exist.

The following example creates a custom connection with a single property:

```
Link connection_link = new Link();
connection_link.setName("designconnection");
connection_link.setType("application/" + Service.class.getName() + "+xml");

Link integer_type = new Link();
integer_type.setName("integer-any");
integer_type.setType("application/" + PropertyType.class.getName() + "+xml");

PropertyDefinition weightPD = new PropertyDefinition();
weightPD.setName("weight");
weightPD.setDisplayName("Weight");
weightPD.setDescription("The weight of the connection");
weightPD.setReadable(true);
weightPD.setWritable(true);
weightPD.setMaxAllowed(1);
weightPD.setPropertyType(integer_type);

AssetType customConnectionType = new AssetType();
customConnectionType.setName("customconnection");
customConnectionType.setDescription("Custom Connection for My Service");
customConnectionType.setDisplayName("Custom Connection");
customConnectionType.setSuperType(connection_link);
customConnectionType.getPropertyDefinitions().add(weightPD);
customConnectionType.setAllowExtensions(true);

RegistrationRequest registration = new RegistrationRequest();
registration.getAssetTypes().add(customConnectionType);
```

Using this new custom connection type when you define a connections for the service is similar to the example in "Defining a Service Connection" on page 68. The difference is that instead of using a link to "`designconnection`" as the connection type, you would use a link to the new "`customconnection`" connection type.

Asset Type Properties

The asset type objects in the service adapter also define the asset type properties needed to create a service instance at deployment, according to the service vendor API. You can specify these properties so that valid values are passed to the service adapter at deploy time. Agility Platform administrators can also manage these properties and modify default values so that the service usage meets their organizational requirements.

Properties are typically those values that are needed to instantiate a service with the provider. For example, this could include an account number, user name, and password for a service provider type. Or, for a service, this could include a protocol and port number.

Required Properties

A service provider might require a property value such that it cannot provision a functional service without it being defined. In this case, the set the minimum number of defined values (`MinRequired`) to a value greater than its default of zero, such as the following example:

```
PropertyDefinition requiredPD = new PropertyDefinition();
requiredPD.setName("required-property");
requiredPD.setDisplayName("Required Property");
requiredPD.setDescription("Example of a required property");
requiredPD.setReadable(true);
requiredPD.setWritable(true);
requiredPD.setMinRequired(1);
requiredPD.setMaxAllowed(1);
requiredPD.setPropertyType(stringType);
```

Property Defaults

A default value is useful where there is a typical standard or format that the consumer of the service will most likely use. Use the `getDefaultValues()` method to specify one or more default values for a property, such as the following example:

```
PropertyDefinition dfltStringPD = new PropertyDefinition();
```

```
dfltStringPD.setName("default-string-property");

dfltStringPD.setDisplayName("Default Valued String");

dfltStringPD.setDescription("Example of a string with a default value");

dfltStringPD.setReadable(true);

dfltStringPD.setWritable(true);

dfltStringPD.setMaxAllowed(1);

dfltStringPD.setPropertyType(stringType);


AssetProperty defaultValue = new AssetProperty();

defaultValue.setStringValue("hello, world");

dfltStringPD.getDefaultValues().add(defaultValue);
```

Property Constraints

In the Agility Platform, property types define the primitive type (string, integer, numeric, Boolean, or date) and optionally specify a pre-defined set of values for one or more properties or script variables. Users can then assign values according to the property types assigned to the properties or variables.

When to constrain the values for a defined property, create a property type to specify the list of valid values, such as the following example:

```
PropertyTypeValue abcValue = new PropertyTypeValue();

abcValue.setName("ABC");

abcValue.setDisplayName(abcValue.getName());

abcValue.setValue(abcValue.getName());


PropertyTypeValue defValue = new PropertyTypeValue();

defValue.setName("DEF");

defValue.setDisplayName(defValue.getName());

defValue.setValue(defValue.getName());


PropertyTypeValue ghiValue = new PropertyTypeValue();

ghiValue.setName("GHI");

ghiValue.setDisplayName(ghiValue.getName());

ghiValue.setValue(ghiValue.getName());


PropertyType listType = new PropertyType();

listType.setName("example-list-type");

listType.setType(PrimitiveType.STRING);

listType.setDisplayName("Example List Constraint");
```

```
listType.setValueConstraint(ValueConstraintType.LIST);

listType.getRootValues().add(abcValue);

listType.getRootValues().add(defValue);

listType.getRootValues().add(ghiValue);


Link listLink = new Link();

listLink.setName(listType.getName());

listLink.setType("application/" + PropertyType.class.getName() + "+xml");


PropertyDefinition listPD = new PropertyDefinition();

listPD.setName("example-list-property");

listPD.setDisplayName("Example List Property");

listPD.setDescription("Example of a property constrained by a list");

listPD.setReadable(true);

listPD.setWritable(true);

listPD.setMaxAllowed(1);

listPD.setPropertyType(listLink);

listPD.setPropertyTypeValue(listType);
```

Property Validators

Properties for a `ServiceProviderType` or `Service` asset type can be validated through the `com.servicemesh.agility.api` validators. These validators are derived from the FieldValidator base class, which has no attributes.

Class	Purpose
DateValidator	Constrains a date value by one or more of "Past", "Future", and "Current"
EmailValidator	Constrains a string value to a valid email address
IntegerRangeValidator	Constrains an integer value by one or more min-max ranges
IntegerScaleValidator	Constrains an integer value to a maximum number of digits
NumericRangeValidator	Constrains a floating point value by one or more min-max ranges
NumericScaleValidator	Constrains a floating point value to a maximum number of digits
RegexValidator	Constrains a string value to match a regular expression pattern
StringLengthValidator	Constrains a string value to a maximum number of characters

String Properties with Validators

The FieldValidator classes can be combined. The following example defines a string property with both length and regular expression validation:

```
PropertyDefinition validStringPD = new PropertyDefinition();

validStringPD.setName("validated-string-property");

validStringPD.setDisplayName("Validates String");

validStringPD.setDescription("Example of a validated string");

validStringPD.setReadable(true);

validStringPD.setWritable(true);

validStringPD.setMaxAllowed(1);

validStringPD.setPropertyType(stringType);


StringLengthValidator lengthValidator = new StringLengthValidator();

lengthValidator.setMaxLength(256);


RegexValidator regexValidator = new RegexValidator();

regexValidator.getExpressions().add("^[A-Za-z0-9-]*$");


validStringPD.setValidator(new FieldValidators());

validStringPD.getValidator().getValidators().add(lengthValidator);

validStringPD.getValidator().getValidators().add(regexValidator);
```

Integer Properties with Validators

The following example includes a range validator to constrains the property values within known range(s):

```
Link integerType = new Link();

integerType.setName("integer-any");

integerType.setType("application/" + PropertyType.class.getName() + "+xml");


PropertyDefinition rangePD = new PropertyDefinition();

rangePD.setName("ranged-property");

rangePD.setDisplayName("Ranged Property");

rangePD.setDescription("Example of an integer property with two allowed ranges");

rangePD.setReadable(true);

rangePD.setWritable(false);
```

```
rangePD.setMaxAllowed(1);
rangePD.setPropertyType(integerType);


IntegerRange rangeOne = new IntegerRange();
rangeOne.setMin("0");
rangeOne.setMin("20");


IntegerRange rangeTwo = new IntegerRange();
rangeTwo.setMin("90");
rangeTwo.setMin("100");


IntegerRangeValidator rangeValidator = new IntegerRangeValidator();
rangeValidator.getRanges().add(rangeOne);
rangeValidator.getRanges().add(rangeTwo);
rangePD.setValidator(new FieldValidators());
rangePD.getValidator().getValidators().add(rangeValidator);
```

The following example includes a scale validator to constrains the property values to a maximum according to the number of digits:

```
PropertyDefinition scalePD = new PropertyDefinition();
scalePD.setName("scaled-property");
scalePD.setDisplayName("Scaled Property");
scalePD.setDescription("Example of an integer property with max 3 digits");
scalePD.setReadable(true);
scalePD.setWritable(false);
scalePD.setMaxAllowed(1);
scalePD.setPropertyType(integerType);


IntegerScaleValidator scaleValidator = new IntegerScaleValidator();
scaleValidator.setMaxDigits(3);
scalePD.setValidator(new FieldValidators());
scalePD.getValidator().getValidators().add(scaleValidator);
```

Service Adapter File Packaging and Versioning

In the course of planning and developing a Agility Platform service adapter, you will want to package it in a way that supports versioning and updates. Good versioning practices will enhance your ability to support the adapter and its use by consumers. With Red Hat Enterprise Linux (RHEL) being the supported basis for a standard Agility Platform installation, this document focuses on the RPM as the packaging mechanism for service adapters.

The Agility Platform convention for versioning components includes these aspects:

- Major version number (A)
- Minor version number (B)
- Build version number (C)
- A count of the number of source code commits in the last build of the software (D)
- Hash value representing the latest source code commit in the last build of the software (E)

Note: If you make significant changes to your service adapter, you will want to increment either the major version number or minor version number to distinguish it from previously released versions.

Service Adapter RPM File Conventions

The versioning aspects are reflected in the RPM file name for a service adapter using the following convention:

```
agility-adapters-azure-trafficmanager-<A>.<B>-<C>.r<D>.<E>.noarch.rpm
```

These aspects are also reflected in the RPM specification:

```
Version: <A>.<B>
Release: <C>.r<D>.<E>
```

The specification provides a mechanism for producing information about the adapter RPM, such as the following:

```
$ rpm -qi agility-adapters-azure-trafficmanager-1.0-0.r91.6288ee6
Name : agility-adapters-azure-trafficmanager Relocations: (not relocatable)
Version : 1.0 Vendor: CSC Agility Dev
```

```
Release : 0.r91.6288ee6 Build Date: Thu 16 Apr 2015 12:35:38 PM CDT

Install Date: Thu 16 Apr 2015 02:44:11 PM CDT Build Host: localhost.localdomain

Group : Services/Cloud Source RPM: agility-adapters-azure-trafficmanager-1.0-
0.r91.6288ee6.src.rpm

Size : 439509 License: Commercial

Signature : (none)

URL : http://www.csc.com/

Summary : Agility Service Pack for Azure Traffic Manager

Description : Service pack adapter between Agility Platform and Microsoft Azure
Traffic Manager.
```

Service Adapter Bundle Conventions

With OSGi as the backbone container for Agility Platform installations, version data also plays a role in the OSGi bundle definition. The Agility Platform convention is:

```
Bundle-Version: <A>.<B>.<C>
```

For support purposes, it is also a best practice to include <D> and <E> within the Bundle-Description text, such as the following:

```
karaf@root()> headers 237 Agility - Azure Traffic Manger Service Provider (237)
-----------------------------------------------------
...
Bundle-Description = Agility adapter to the Microsoft Azure Traffic Manager
service. Revision r91.6288ee6.
Bundle-SymbolicName =
com.servicemesh.agility.adapters.service.azure.trafficmanager
Bundle-Version = 1.0.0
...
```

Another Agility Platform file convention is the name of the jar file containing a service adapter:

```
<package-name>-<A>.<B>.<C>.jar
```

The following is an example of this convention:

```
$ rpm -ql agility-adapters-azure-trafficmanager-1.0-0.r91.6288ee6
/opt/agility-
platform/deploy/com.servicemesh.agility.adapters.service.azure.trafficmanager-
1.0.0.jar
```

Building a Service Adapter with Versioning

There are many ways to build versioning information into an adapter RPM and OSGi manifest. This section provides an example where versioning components are specified once in the Ant build.xml file and then propagated at build time.

Mapping the versioning components

The first task is to map the version components into Ant constructs using the versioning identifiers, such as the following:

Version parameter	Ant property
<A> - Major version number	<property name="version.major" value="1" />
<B> - Minor version number	<property name="version.minor" value="0" />
<C> - Build version number	<property name="version.build" value="0" />
<D> - Source commit count	<exec executable="bash" outputproperty="git.revision"> <arg value="-c" /> <arg value="git rev-list HEAD | wc -l | sed 's/^ *//'" /> </exec>
<E> - Hash of latest source commit	<exec executable="bash" outputproperty="git.commit"> <arg value="-c" /> <arg value="git log -n1 --pretty=format:%h HEAD" /> </exec>

You can then collate the five properties for easier use, such as the following:

```
<property name="bundle.version"
value="${version.major}.${version.minor}.${version.build}" />

<property name="rpm.version" value="${version.major}.${version.minor}-
${version.build}"/>

<property name="git.summary" value="${git.revision}.${git.commit}"/>
```

Populating the OSGi bundle information

To populate OSGi bundle information at runtime, use a template for the manifest that includes the following:

```
Bundle-Version: %bundle_version
```

Then copy and update the manifest via Ant, similar to the following:

```
<property name="manifest" value="${bin}/META-INF/MANIFEST.MF"/>
<copy file="META-INF/MANIFEST.MF" tofile="${manifest}" overwrite="true"/>
<replace file="${manifest}" token="%bundle_version" value="${bundle.version}"/>
<echo file="${manifest}" append="true"
message="Bundle-Description: Agility adapter to the Microsoft Azure Traffic
Manager service. Revision r${git.summary}.${line.separator}"/>
```

Populating the resource properties

Include versioning information in a resource properties file for consumption by the adapter code, similar to the following:

```
<property name="propfile" value="${bin}
/resources/TrafficManagerAdapter.properties"/>
<echo file="${propfile}">adapter.vendor=CSC Agility Dev
adapter.version=${bundle.version}
adapter.revision=r${git.summary}
</echo>
```

The manifest and properties file are included in a jar that meets the Agility Platform naming convention of "<package-name>-<A>.<B>.<C>.jar", such as the following:

```
<property name="bundle.name" value="${ant.project.name}-${bundle.version}"/>
<property name="bundle.jar" value="${bundle.name}.jar"/>
<property name="bin.bundle.jar" value = "${bin}/${bundle.jar}"/>

<jar destfile="${bin.bundle.jar}" manifest="${manifest}" basedir="${bin}">
    <fileset dir="${bin}">
        <include name="**/*.class"/>
        <include name="**/*.properties"/>
    </fileset>
    <fileset dir="${lib}" includes="*.jar"/>
</jar>
```

Populating the RPM at runtime

To populate the RPM information at runtime, the specification file includes the following substitutions:

```
Version: %rpm_version
Release: %rpm_revision
```

Then you generate the RPM from Ant, similar to the following:

```
<property name="rpm.basename" value="agility-adapters-azure-trafficmanager"/>
<property name="rpm.name" value="${rpm.basename}-${rpm.version}"/>

    <resources id="rpm.args">
        <string>-bb</string>
        <string>--define '_topdir ${basedir}/${rpm}'</string>
        <!-- Agility convention is to only put major.minor as RPM version -->
        <string>--define 'rpm_version ${version.major}.${version.minor}'</string>
        <string>--define 'rpm_revision ${version.build}.r${git.summary}'</string>
        <string>--buildroot ${basedir}/${buildroot}</string>
    </resources>
<pathconvert property="rpm.command" refid="rpm.args" pathsep=" "/>
<rpm specFile="${rpm.basename}.spec" topDir="${rpm}" failOnError="true"
command="${rpm.command}"/>
```

Service adapter version update

A service adapter will typically change either its major version number or minor version number when it is updated to a newer version. The build version number can remain at zero if the developer follows the pattern of generating source code control data at build time and including it in RPM and bundle manifest.

This update is a simple process when you use the Ant properties to drive versioning. For example, to move to version 3.2.0, edit the build.xml file as follows:

```
<property name="version.major" value="3" />
<property name="version.minor" value="2" />
```

Then you can commit the change and do a clean rebuild.

Service adapters implement a series of interfaces to transform Agility Platform service operations into service-specific operations. This chapter provides information about the potential interfaces that a service adapter can implement. Service adapters must implement some of these interfaces while others are optional. The registration message created by the Service adapter tells the Services framework which optional interfaces the adapter implements.

The following ERD provides an overview of the relevant entities within the Agility Platform data model.
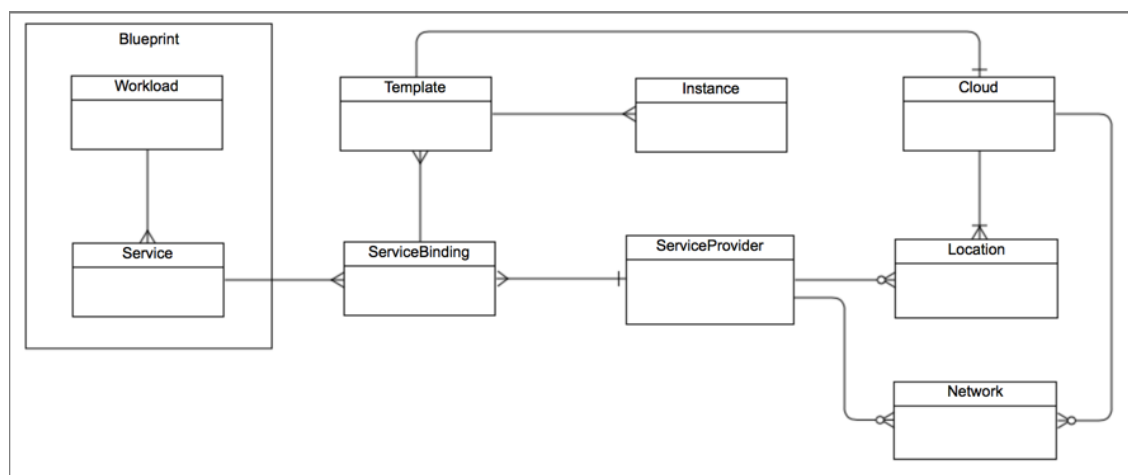
Figure 7 - Data entities supporting application services

Upon deployment of the blueprint, Agility Platform creates a service binding (ServiceBinding) to map the service requirements/configuration (Service) to a specific service provider (ServiceProvider). Service providers can be global in scope or optionally constrained to specific cloud environments/locations. The service binding is passed to the service adapter along with the desired service configuration (service asset meta-model properties) on deployment to instantiate an actual service instance (e.g. database account, virtual load balancer instance, etc). This service binding can be used by the adapter to persist the state of the service instance. Lifecycle operations are invoked on the service adapter as instances are

provisioned, started, stopped, released to allow the adapter to inject configuration into the workload and/or make changes to the external service (e.g. register the instance with a load balancer) as required.

The Reactor-Promise Design Pattern

The Agility Platform Services SDK incorporates the Reactor design pattern (http://en.wikipedia.org/wiki/Reactor_pattern) as provided by the Core Frameworks bundle (`com.servicemesh.core`). A key class for using the Agility PlatformReactor is `com.servicemesh.core.async.Promise`, which contains the results of a work request overseen by the Reactor.

`Promise` provides a functional approach for a service adapter to conduct the asynchronous operations defined by the Services SDK operations interfaces. The basis for this functional programming paradigm is the `Function` interface:

```
package com.servicemesh.core.async;

public interface Function<A,R> {
    public R invoke(A arg);
}
```

Class hierarchy

The following represents the class hierarchy for `Promise`:

```
Promise<T>
    |
    v
CompletablePromise<T>
```

`CompletablePromise` is not typically utilized directly in service adapter code.

Promise Status Methods

Use the following methods to retrieve status information for a `Promise`:

Method	Type	Description
isCancelled()	Boolean	Returns true if the request was canceled
isCompleted()	Boolean	Returns true if the request completed
isFailed()	Boolean	Returns true if the request completed with a failure

You can use the return values for these methods in combination to determine specific information about a `Promise`:

State	isCancelled()	isCompleted()	isFailed()
In progress	false	false	false
Completed after a failure	false	false	true
Completed successfully	false	true	false
Canceled while in progress	true	false	false

Promise Control Methods

Use the following callback methods for a `Promise`:

Method	Type	Description
cancel()	void	Signals that the promise is canceled and notifies any pending threads or pending results
onCancel(final Callback<Void> cb)	void	Registers a callback to be invoked on cancellation of the promise The callback is executed immediately if the promise is already canceled.

Method	Type	Description
onComplete(final Callback<T> cb)	void	Registers a callback to be invoked on completion of the promise The callback is executed immediately if the promise is already completed.
onFailure(final Callback<T> cb)	void	Registers a callback to be invoked if a failure condition occurs The callback is executed immediately if the promise is already failed.
recover(final Function<Throwable, T> func)	Promise<T>	Wrap this promise with a promise that will handle exceptions throws.

A service adapter can also choose to register callback methods for a canceled, completed, and/or failed Promise, such as the following:

```
package com.servicemesh.core.async;

public interface Callback<T> {
    public void invoke(T arg);
}
```

For example, you can cancel an in-progress Promise using the `cancel()` method and then use the `recover()` method to convert an exception condition into a default value of the expected type, such as returning zero for an Integer operation where illegal arithmetic is attempted.

```
final Promise<Integer> promise2 = promise.recover(new Function<Throwable, Integer>
() {
    @Override
    public Integer invoke(final Throwable th)
    {
        if (th instanceof ArithmeticException) {
            return 0;
        }
        throw new IllegalArgumentException();
    }
});
```

Promise Compositional Methods

Use the following compositional methods for a `Promise`:

Method	Type	Description
flatMap(final Function<T, Promise<R>> func)	<R> Promise<R>	Returns a new promise that, on completion of the current promise, completes by mapping the result using the supplied function
map(final Function<T, R> func)	<R> Promise<R>	Returns a new promise that, on completion of the current promise, completes by mapping the result using the supplied function
pure(final T result)	static <T> Promise<T>	Returns a completed promise with the specified result
pure(Throwable th)	static <T> Promise<T>	Returns a completed promise with the specified error result
sequence(final List<Promise<T>> promises)	static <T> Promise<T>	Returns a promise that completes when the supplied list of promises complete
sequenceAny(List<Promise<?>> promises)	static Promise<List<Object>>	Returns a promise that completes when the supplied list of promises complete

pure()

The `pure()` methods return a direct conclusion, which is the simplest type of result:

```
try {
    ServiceProviderResponse response = new ServiceProviderResponse();
    response.setStatus(Status.COMPLETE);
    // This Promise's isCompleted() will return true
    return Promise.pure(response);
}
catch (Exception e) {
    String err = "Exception for " + method.getName() + "'" + uri +
        "': " + e.toString();
    // This Promise's isFailed() will return true
    return Promise.pure(new Exception(err));
}
```

Map()

A service adapter typically uses `map()` to translate between an external cloud vendor data type and an internal Agility Platform data type. In the following example, the internal type is `java.lang.Integer`:

```
final Promise<Integer> mapped = promise.map(new Function<CloudType, Integer>() {

    @Override

    public Integer invoke(final CloudType result)

    {

        // If no exception is thrown, the returned Promise's isCompleted()

        // will return true.

        return new Integer(result.asInteger());

    }

});
```

FlatMap()

The `flatMap()` method can translate from a set of intermediate results to a promise with the final aggregated result, such as the following example:

```
final Promise<Integer> flattened = promise.flatMap(new Function<List<CloudType>,
Promise<Integer>>() {

    @Override

    public Promise<Integer> invoke(final List<CloudType> intermediate)

    {

        int sum = 0;

        for (CloudType result : intermediate) {

        sum += result.asInteger();

        }

        return Promise.pure(new Integer(sum));

    }

});
```

Sequence()

The `sequence()` method, together with a mapping method, can combine the results of multiple Promises after they are completed, such as the following example:

```
Promise<Integer> mappedGrandTotal(List< Promise<Integer> > promises)

{

    Promise< List<Integer> > promise = Promise.sequence(promises);
```

```
    return promise.map(new Function<List<Integer>, Integer>() {
        @Override
        public Integer invoke(final List<Integer> subs)
        {
            int grand = 0;
            for (Integer sub : subs) {
            grand += sub.intValue();
            }
            return new Integer(grand);
        }
    });
}
```

SequenceAny()

The `sequenceAny()` method is similar to `sequence()` but can work with Promises associated with generic types. The following is an example:

```
Promise<Integer> flattenedGrandTotal(List< Promise<Object> > promises)
{
    Promise< List<Object> > promise = Promise.sequence(promises);

    return promise.flatMap(new Function< List<Object> , Promise<Integer> >() {
        @Override
        public Promise<Integer> invoke(final List<Object> subs)
        {
            int grand = 0;
            for (Object sub : subs) {
                if (sub instanceof Integer) {
                grand += ((Integer)sub).intValue();
                }
                else if (sub instanceof Long) {
                grand += ((Long)sub).intValue();
                }
                else if (sub instanceof String) {
                    try {
                    grand += Integer.parseInt((String)sub);
                    }
                    catch (NumberFormatException nfe) {
                    String err = "Invalid sub: " + (String)sub;
                    return Promise.pure(new Exception(err));
```

```
                    }
                }
            }
            return Promise.pure(new Integer(grand));
        }
    });
}
```

The CompletablePromise Class

`CompletablePromise` has two public methods that are invoked when an operation is performed by the Agility Platform Reactor. The methods are also invoked by the base Promise `pure()` methods.

Method	Type	Description
complete(final T result)	void	Signals that this promise is complete and notify any pending threads or mapped results
failure(final Throwable th)	void	Signals that this promise is completed with an error and notify any pending threads or mapped results

A unit test for adapter code can utilize a `CompletablePromise` that is created via the `PromiseFactory`, such as the following:

```
import com.servicemesh.core.async.PromiseFactory;

CompletablePromise<Integer> promise = PromiseFactory.create();
promise.complete(100);

CompletablePromise<Integer> promise2 = PromiseFactory.create();
promise2.failure(new Exception("Test"));
```

The `PromiseFactory` simply returns a `DefaultCompletablePromise`, such as the following:

```
public class PromiseFactory
{
    public static <T> CompletablePromise<T> create()
    {
        return new DefaultCompletablePromise<T>();
    }
}
```

HttpClient with Promises

Use the Async HttpClient library to easily execute HTTP requests and asynchronously process the HTTP responses. `HttpClient` has one public method promise, which takes in a request of type `IHttpRequest` and returns a Promise of type `IHttpResponse`.

Method	Type	Description
promise(final IHttpRequest request)	Promise<IHttpResponse>	Executes a request of type IHttpRequest and returns a Promise of type IHttpResponse

The Promise call takes an HTTP request and returns a promise with a value of type `IHttpResponse`, such as the following:!

```
String uri = new String("http://www.csc.com");

IHttpRequest request = HttpClientFactory.getInstance().createRequest
(HttpMethod.GET, uri);

Promise<IHttpResponse> promise = httpClient.promise(request);


IHttpResponse response = promise.get()
```
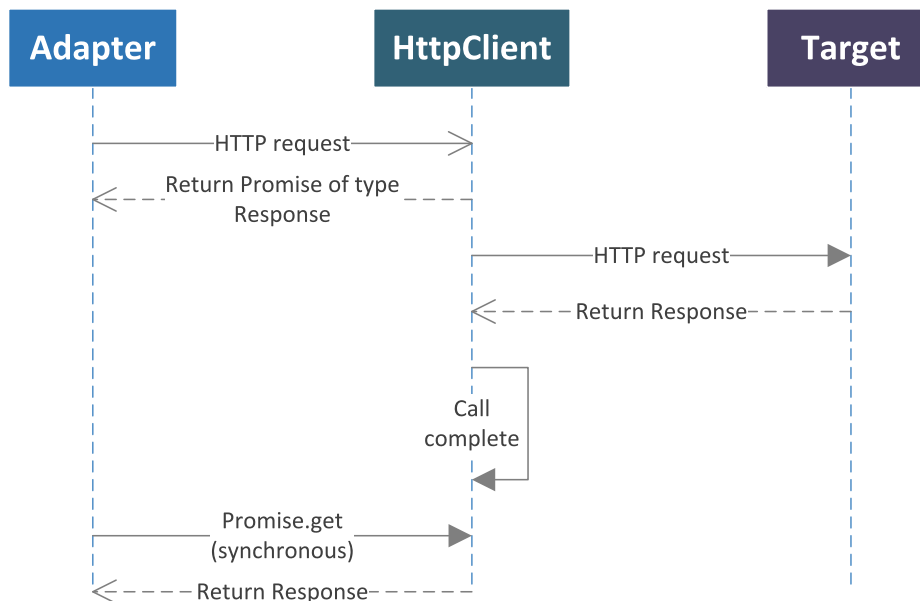

Figure 8 - Message flow for promise

The following is an example of a Promise with callback:

```
String uri = new String("http://www.csc.com");

IHttpRequest request = HttpClientFactory.getInstance().createRequest
(HttpMethod.GET, uri);

Promise<IHttpResponse> promise = httpClient.promise(request);

Promise<String> flatPromise = promise.flatMap(new function<IHttpResponse,
Promise<String>(){


   public Promise<R> invoke(IHttpResponse response)

   {

       if (response.getStatusCode() != 200 ){

           return Promise.pure(new String("response error"));


       }

       return Promise.pure(new String(response.getContent()));

   }

}
```
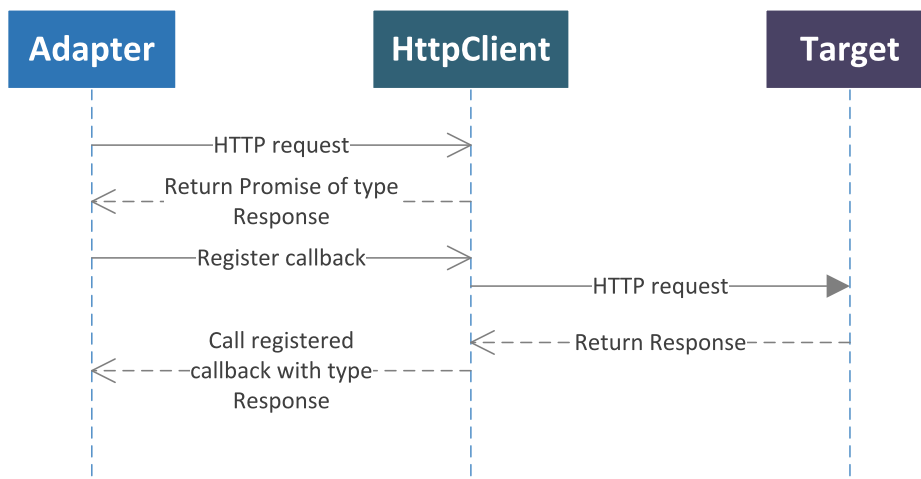

Figure 9 - Message flow using callback

The following is an example of implementation of HTTP methods (Get, Post, Put, and Delete):

```
@Override
public <T> Promise<T> get(String requestURI, QueryParams params,
   final Class<T> responseClass)
{
    return execute(HttpMethod.GET, requestURI, params, null, responseClass);
```

```
}
@Override
public <T> Promise<T> post(String requestURI, Object resource,
   final Class<T> responseClass)
{
    return execute(HttpMethod.POST, requestURI, null, resource, responseClass);
}
@Override
public <T> Promise<T> put(String requestURI, Object resource,
   final Class<T> responseClass)
{
    return execute(HttpMethod.PUT, requestURI, null, resource, responseClass);
}
@Override
public Promise<IHttpResponse> delete(String requestURI)
{
    return execute(HttpMethod.DELETE, requestURI, null, null,
        IHttpResponse.class);
}
<T> Promise<T> execute(HttpMethod method,
    String requestURI,
    QueryParams params,
    Object resource,
    final Class<T> responseClass)
{
    try {
        uri = getURI(requestURI, params);
        IHttpRequest request =
            HttpClientFactory.getInstance().createRequest(method, uri);
        addMsVersionHeader(request);

        if (resource != null) {
            String content;
            if (resource instanceof java.lang.String)
                content = (String)resource;
            else
                content = _endpoint.encode(resource);
            addContentTypeHeader(request);
            request.setContent(content);
        }
        if (_logger.isDebugEnabled()) {
            _logger.debug(method.getName() + " " + uri);
```

```
        }
        Promise<IHttpResponse> promise = _httpClient.promise(request);
        if (responseClass.getCanonicalName()
            .equals(IHttpResponse.class.getCanonicalName()))
            return (Promise<T>)promise;
        else
            return promise.map(new Function<IHttpResponse, T>() {
                @Override
                public T invoke(IHttpResponse response) {
                    return _endpoint.decode(response, responseClass);
                }
            });
    }
    catch (SignatureException ex) {
        String err = "Authorization failed: " + ex.toString();
        return Promise.pure(new Exception(err));
    }
    catch (Exception e) {
        String err = "Exception for " + method.getName() + "'" + uri +
            "': " + e.toString();
        return Promise.pure(new Exception(err));
    }
}
```

ServiceAdapter Class and SPI Interfaces

Every service adapter extends the Services SDK `ServiceAdapter` class. The following diagram illustrates an example where AdapterClass represents the implementing class for a service adapter.

```
org.osgi.framework.BundleActivator
                  △
                  ┆
com.servicemesh.agility.sdk.service.spi.ServiceAdapter
                  △
                  │
             AdapterClass
```
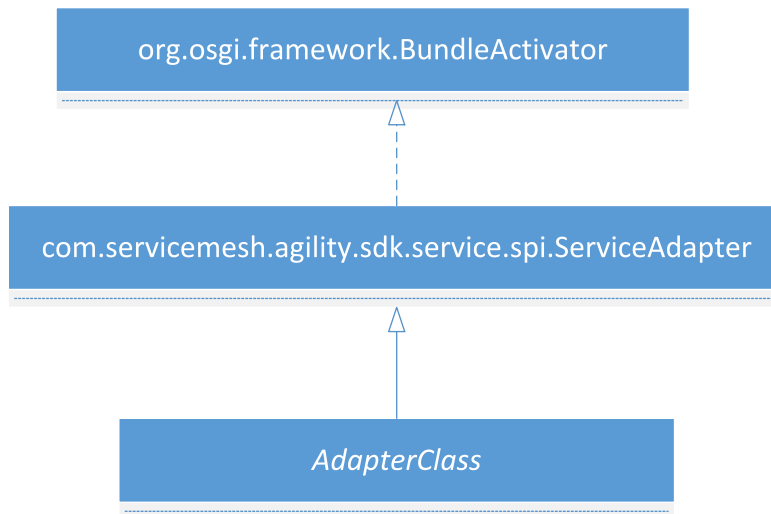
Figure 10 - Extending the ServiceAdapter class

ServiceAdapter Methods

The service adapter class ***must*** implement the following `ServiceAdapter` abstract methods:

Method	Type	Description
getServiceProviderTypes()	List<ServiceProviderType>	Returns the Service Provider Types to be registered as an OSGi service
getRegistrationRequest()	RegistrationRequest	Returns the metadata for the capabilities provided by the service adapter
onRegistration (RegistrationResponse response)	void	Called after adapter registration with the Agility Platform service framework

Method	Type	Description
getServiceProviderOperations()	IServiceProvider	Required interface used to manage the service provider
getServiceInstanceOperations()	IServiceInstance	Required interface used to manage the lifecycle of bindings (service instances) to the service provider

For more information about the RegistrationRequest, see "The RegistrationRequest Class and Object" on page 50.

The service adapter class can optionally choose to override the following `ServiceAdapter` methods:

Method	Type	Description
getInstanceOperations()	IInstanceLifecycle	Implements hooks in instance lifecycle for workloads dependent on the service The default value is null.
getServiceInstanceLifecycleOperations ()	IServiceInstanceLifecycle	Implements hooks in service instance lifecycle for service instances dependent on the service The default value is null.
getConnectionOperations()	IConnection	Manages the lifecycle of connections to the service provider The default value is null.
getAssetNotificationOperations()	IAssetLifecyle	Receives lifecycle (CRUD) notifications for specific asset types The default value is null.

ServiceAdapter Initialization

The initialization sequence for the service adapter results in a notification to the `ServiceFramework` so that it can service as the proxy for later operations involving the service adapter.
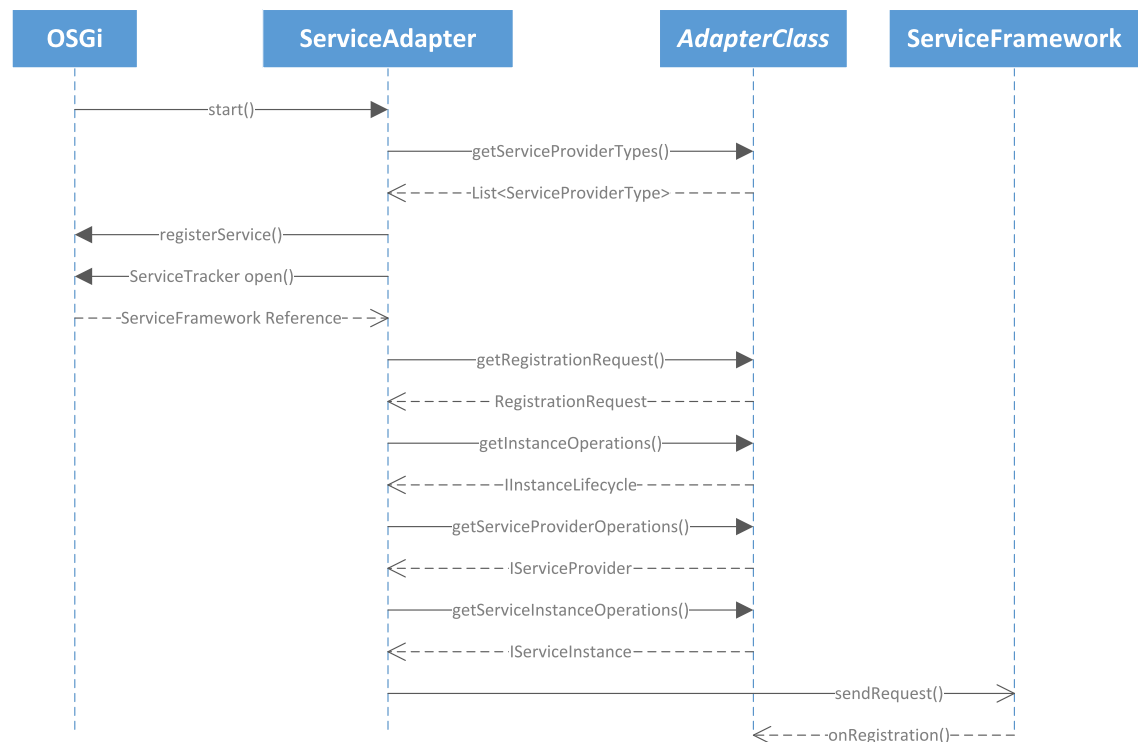
Figure 11 - ServiceAdapter initialization sequence

The `RegistrationResponse` passed into `onRegistration` includes these methods:

Method	Type	Description
getAssetTypes()	List<AssetType>	Returns the persisted Asset Types upon the completion of registration
getServiceProviderTypes()	List<ServiceProviderType>	Returns the persisted Service Provider Types upon the completion of registration

Method	Type	Description
getServiceProviders()	List<ServiceProvider>	Returns any existing Service Providers for the registered Service Provider Types
getClouds()	List<Cloud>	Returns any Clouds associated with existing Service Providers

> *Note:* An empty implementation of `onRegistration` is valid if the adapter has no need to immediately work with the registered Service Provider Type(s).
>
> Because they are only utilized on an as-needed basis for deployed assets associated with the service provided by the service adapter, the `getServiceInstanceLifecycleOperations()`, `getConnectionOperations()`, and `getAssetNotificationOperations()` methods are not called at registration time.

A service adapter class also has access to these `ServiceAdapter` methods:

Method	Type	Description
getServiceRegistry()	ServiceRegistry	Returns a ServiceRegistry object that can be used to look up other Agility Platform services
createAsset(Asset asset, Asset parent)	Promise<Asset>	Returns a promise to create an Asset
getAsset(String assetType, int id)	Promise<Asset>	Returns a promise to retrieve an Asset by its type name and identifier
getAssets(String assetType, List<Property> params)	Promise<List<Asset>>	Returns a promise to retrieve an Asset list by its type name and optional search query parameters
updateAsset(Asset asset, Asset parent)	Promise<Asset>	Returns a promise to update an Asset
deleteAsset(Asset asset, Asset parent)	Promise<Asset>	Returns a promise to delete an Asset

JAXBContext for Vendor API

Many cloud vendors provide an HTTP/REST API for administering a service. As a result, a common pattern for service adapters is the serialization/deserialization of vendor objects via the Java Architecture for XML Binding (JAXB). To effectively run within an OSGi context, it is advisable to retain a JAXBContext for serialization operations.

The following simplified schema provides an illustration of this implementation:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
        xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
        xmlns:example="http://schemas.vendor.com/exampleservice"
        targetNamespace="http://schemas.vendor.com/exampleservice"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        jaxb:extensionBindingPrefixes="xjc"
        jaxb:version="2.0">

    <xsd:annotation>
        <xsd:appinfo>
            <jaxb:globalBindings>
                <xjc:simple />
                <jaxb:serializable uid="1"/>
            </jaxb:globalBindings>
                <jaxb:schemaBindings>
                <jaxb:package name="com.vendor.schemas.example.service"/>
            </jaxb:schemaBindings>
        </xsd:appinfo>
    </xsd:annotation>

    <xsd:element name="ServiceError" type="example:ServiceError"/>
    <xsd:complexType name="ServiceError">
        <xsd:sequence>
            <xsd:element name="Id" type="xsd:integer"/>
            <xsd:element name="Message" type="xsd:string"/>
        </xsd:sequence>
```

```
    </xsd:complexType>

    <xsd:element name="Service" type="example:Service"/>
    <xsd:complexType name="Service">
        <xsd:sequence>
            <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Status" type="example:Status" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="Status">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Up"/>
            <xsd:enumeration value="Down"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

The service adapter generates JAXB classes from this schema at build time. At runtime, the adapter retains a JAXBContext for its serialization operations.

```
import java.io.ByteArrayOutputStream;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import com.vendor.schemas.example.service.ServiceError;
import com.servicemesh.io.http.HttpClientException;
import com.servicemesh.io.http.HttpUtil;
import com.servicemesh.io.http.IHttpResponse;
public class MyEndpoint
{
    private static class ContextHolder
    {
        private static final JAXBContext CONTEXT = getContext();
        private static JAXBContext getContext()
        {
            ClassLoader cl = Thread.currentThread().getContextClassLoader();
            try {
                Thread.currentThread().setContextClassLoader
                (ServiceError.class.getClassLoader());
                return JAXBContext.newInstance(ServiceError.class.getPackage
                ().getName());
            }
```

```
            catch(Exception ex) {                                          99

                logger.error(ex);

                return null;

            }

            finally {

                Thread.currentThread().setContextClassLoader(cl);

            }

        }

    }

    public <T> T deserialize(IHttpResponse response, Class<T> responseClass)

    {

        if (responseClass.isInstance(response))

            return responseClass.cast(response);

        Object object = HttpUtil.decodeObject(response.getContent(), null,
        ContextHolder.CONTEXT);

        if(responseClass.isInstance(object)) {

            return responseClass.cast(object);

        }

        else if (object instanceof ServiceError) {

            ServiceError errorResponse = (ServiceError)object;

            throw new RuntimeException(errorResponse.getMessage());

        }

        throw new RuntimeException("Unable to decode response");

    }

    public String serialize(Object obj)

    {

        ByteArrayOutputStream os = new ByteArrayOutputStream();

        try {

            Marshaller marshaller = ContextHolder.CONTEXT.createMarshaller();

            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

            marshaller.marshal(obj, os);

            return os.toString();

        }

        catch (Exception ex) {

            throw new RuntimeException("Unable to encode object: " + ex.getMessage
            ());

        }

    }

}
```

Service Provider (IServiceProvider)

The `com.servicemesh.agility.sdk.service.spi.IServiceProvider` interface returned by `getServiceProviderOperations()` includes a standard set of operations for managing the service provider, as well as lifecycle callouts for changes to the service provider configuration/definition.

These methods operate within the `Promise<ServiceProviderResponse>` type:

Method	Description
preCreate(ServiceProviderPreCreateRequest request)	Used by the service provider to validate the configuration of the adapter and allow/reject the settings
postCreate (ServiceProviderPostCreateRequest request)	Called after successful configuration of the service provider A service adapter could require some sync of the initial configuration with the actual service provider.
preUpdate (ServiceProviderPreUpdateRequest request)	Used by the service adapter to validate the configuration of the service provider and allow/reject the settings
postUpdate (ServiceProviderPostUpdateRequest request)	Called after successful configuration of the service provider A service adapter could require some sync of the new configuration with the actual service provider.
preDelete(ServiceProviderPreDeleteRequest request)	Used by the service provider to validate the system is in a valid state to perform a delete
postDelete (ServiceProviderPostDeleteRequest request)	Used to clean up any resources associated with the service provider instance
ping(ServiceProviderPingRequest request)	Called by Agility Platform to validate connectivity with the service provider
sync(ServiceProviderSyncRequest request)	Called by Agility Platform to request a sync of the current service configuration with the provider

Method	Description
start(ServiceProviderStartRequest request)	Called from Agility Platform to request a start of the service provider ***Note:*** This might not apply to many service providers. In this case, the service adapter should simply indicate successful completion of the request.
stop(ServiceProviderStopRequest request)	Called from Agility Platform to request a stop of the service provider ***Note:*** This might not apply to many service providers. In this case, the service adapter should simply indicate successful completion of the request.

ServiceProviderRequest Class

The Services SDK `ServiceProviderRequest` class is the basis for the parameter to the IServiceProvider methods. The `ServiceProviderRequest` methods provide information to the service adapter:

Method	Type	Description
getProvider()	ServiceProvider	Returns the service provider
getSettings()	List<Property>	Returns zero or more configuration Property objects as specified by the service adapter RegistrationRequest
getClouds()	List<Cloud>	Returns the cloud(s) associated with the service provider
getProxies()	List<Proxy>	Returns zero or more proxies used to configure communications with the service provider
getLocations()	List<Location>	Returns zero or more locations associated with the service provider
getNetworks()	List<Network>	Returns zero or more networks associated with the service provider
getServiceProviderTypes()	List<ServiceProviderType>	Returns a list of all service provider types used by assets associated with a provided service
getServiceProviders()	List<ServiceProvider>	Returns a list of all service providers used by assets associated with a provided service

> **Note:** The `ServiceProviderType` and `ServiceProvider` lists for associated assets is passed to a service adapter so that it can optionally collaborate with them during an operation.

ServiceProviderResponse Class

The Services SDK `ServiceProviderResponse` class is returned by an `IServiceProvider` method. The `ServiceProviderResponse` object allows a service adapter to provide information back to the Agility Platform Services framework.

Method	Type	Description
getModified()	List<Asset>	Returns the Agility Platform asset(s) with attributes modified by the service adapter
		The service framework subsequently persists the assets.

ServiceProviderOperations Methods

Most of the `IServiceProvider` methods use Services SDK derivations of `ServiceProviderRequest` that provide no additional attributes:

- `ServiceProviderPreCreateRequest`
- `ServiceProviderPostCreateRequest`
- `ServiceProviderPreUpdateRequest`
- `ServiceProviderPostUpdateRequest`
- `ServiceProviderPreDeleteRequest`
- `ServiceProviderPostDeleteRequest`
- `ServiceProviderPingRequest`
- `ServiceProviderStartRequest`
- `ServiceProviderStopRequest`

`ServiceProviderSyncRequest` provides the following additional information:

Method	Type	Description
getServiceInstances()	List<ServiceInstance>	Returns a list of all service instances associated with the service provider
getConnectedAssets()	List<Asset>	Returns a list of all assets connected to the service instances
getConnections()	List<Connection>	Returns a list of all connections to the service instances

The Services SDK provides
`com.servicemesh.agility.sdk.service.operations.ServiceProviderOperations,`
a default implementation of `IServiceProvider` for which every method simply returns a completed promise:

```
ServiceProviderResponse response = new ServiceProviderResponse();
response.setStatus(Status.COMPLETE);
return Promise.pure(response);
```

A service adapter should extend `ServiceProviderOperations` and override methods as needed. For example, implementations of `preCreate()` and `preUpdate()` could verify that the required access credentials are available for a service provider while an adapter `ping()` could attempt an actual connection using those credentials, such as the following example:

```
import java.net.URI;
import java.util.List;
import com.servicemesh.agility.sdk.service.msgs.ServiceProviderPingRequest;
import com.servicemesh.agility.sdk.service.msgs.ServiceProviderPreCreateRequest;
import com.servicemesh.agility.sdk.service.msgs.ServiceProviderPreUpdateRequest;
import com.servicemesh.agility.sdk.service.msgs.ServiceProviderRequest;
import com.servicemesh.agility.sdk.service.msgs.ServiceProviderResponse;
import com.servicemesh.agility.api.Cloud;
import com.servicemesh.agility.api.Credential;
import com.servicemesh.agility.api.Property;
import com.servicemesh.agility.api.ServiceProvider;
import com.servicemesh.core.async.Function;
import com.servicemesh.core.async.Promise;
import com.servicemesh.core.messaging.Status;
import com.servicemesh.io.http.Credentials;
```

```
import com.servicemesh.io.http.HttpClientFactory;
import com.servicemesh.io.http.HttpMethod;
import com.servicemesh.io.http.IHttpClient;
import com.servicemesh.io.http.IHttpClientConfigBuilder;
import com.servicemesh.io.http.IHttpRequest;
import com.servicemesh.io.http.IHttpResponse;

public class MyProviderOperations extends ServiceProviderOperations
{
    @Override
    public Promise<ServiceProviderResponse> preCreate
    (ServiceProviderPreCreateRequest request)
    {
        return Promise.pure(verifyCredentials(request));
    }
    @Override
    public Promise<ServiceProviderResponse> preUpdate
    (ServiceProviderPreUpdateRequest request)
    {
        return Promise.pure(verifyCredentials(request));
    }
    private ServiceProviderResponse verifyCredentials
    (ServiceProviderPreCreateRequest request)
    {
        ServiceProviderResponse response = new ServiceProviderResponse();
        Credential cred = getCredentials(request.getProvider(), request.getClouds
        ());
        if (cred != null) {
            response.setStatus(Status.COMPLETE);
        }
        else {
            response.setStatus(Status.FAILURE);
            response.setMessage("No credentials");
        }
        return response;
    }
    @Override
    public Promise<ServiceProviderResponse> ping(ServiceProviderPingRequest
    request)
    {
```

```
Credential cred = getCredentials(request.getProvider(), request.getClouds
());
if (cred == null) {
    ServiceProviderResponse response = new ServiceProviderResponse();
    response.setStatus(Status.FAILURE);
    response.setMessage("No credentials");
    return Promise.pure(response);
}
Credentials ioCreds = new Credentials
(Credentials.CredentialsType.CREDENTIALS_TYPE_USERNAMEPASSORD);
ioCreds.setUsername(cred.getPublicKey());
ioCreds.setPassword(cred.getPrivateKey());

List<Property> settings = request.getSettings();

IHttpClientConfigBuilder cb = HttpClientFactory.getInstance
().getConfigBuilder();
cb.setCredentials(ioCreds);
cb.setConnectionTimeout(Config.getHttpTimeout(settings));
cb.setRetries(Config.getHttpRetries(settings));
cb.setSocketTimeout(Config.getSocketTimeout(settings));

IHttpClient client = HttpClientFactory.getInstance().getClient(cb.build
());

URI uri = new URI("http://" + Config.getHost(settings) + "/status");
IHttpRequest httpRequest =
    HttpClientFactory.getInstance().createRequest(HttpMethod.GET, uri);

Promise<IHttpResponse> promise = client.promise(httpRequest);
return promise.map(
    new Function<IHttpResponse, ServiceProviderResponse>() {
    @Override
        public ServiceProviderResponse invoke(IHttpResponse arg)
        {
            ServiceProviderResponse response =
                new ServiceProviderResponse();
            if (arg.getStatusCode() == 200) {
                response.setStatus(Status.COMPLETE);
            }
            else {
                response.setStatus(Status.FAILURE);
```

```
                    response.setMessage("ping failed");
            }
            return response;
        }
    });
    }
}
```

Instance Lifecycle (IInstanceLifecycle)

A Blueprint can contain design connections between a Workload and a Service. When the Blueprint is deployed, Agility Platform creates an Instance asset for each Workload and a ServiceInstance asset for each Service. The design connections define the dependencies between these created assets:

Connection source	Connection destination	Description
Instance	ServiceInstance	Instance is dependent on the ServiceInstance. For example, ServiceInstance might represent a load balancing service that will balance traffic between the Instance and other connected VM Instances.
ServiceInstance	Instance	ServiceInstance is dependent on the Instance. For example, ServiceInstance might represent a data service that requires data from the Instance and other connected VM Instances.

The Service SDK `IInstanceLifecycle` interface represents the operations for a service adapter to consider when an Instance bound to a ServiceInstance changes state.

These methods operate within the `Promise<InstanceResponse>` type:

Method	Description
preProvision(InstancePreProvisionRequest request)	Indicates beginning of the provisioning workflow for a connected Instance ***Note:*** The service adapter has the option to abort the provisioning if it cannot support another VM.
postProvision(InstancePostProvisionRequest request)	Indicates completion of the provisioning workflow for a connected Instance ***Note:*** The service adapter has the option to register the VM with a service offering and/or allocate service resources for the VM.
preBoot(InstancePreBootRequest request)	Indicates the VM for a connected Instance is preparing to boot (cloned but not powered) ***Note:*** This provides a hook for a networking service adapter to allocate network resources (such as an IP address) for the VM.
postBoot(InstancePostBootRequest request)	Indicates the VM for a connected Instance has booted In this state, it should have an IP address but configuration management is not yet applied.
preStop(InstancePreStopRequest request)	Indicates the VM for a connected Instance is preparing to stop
postStop(InstancePostStopRequest request)	Indicates the VM for a connected Instance is stopped without releasing its underlying resources ***Note:*** The service adapter has the option to deregister/cleanup resources associated with the stopped VM.
preStart(InstancePreStartRequest request)	Indicates the VM for a connected Instance that is in a stopped state is preparing to start ***Note:*** The service adapter has the option to verify that the VM settings are compatible with the service.
postStart(InstancePostStartRequest request)	Indicates the startup of the VM for a connected Instance is completed ***Note:*** The service adapter has the option to register or provision service resources for the VM.

Method	Description
preRestart(InstancePreRestartRequest request)	Indicates the VM for a connected Instance is preparing to restart (similar to the power-cycling of a physical hardware server)
postRestart(InstancePostRestartRequest request)	Indicates the restart of the VM for a connected Instance is completed ***Note:*** The service adapter has the option to register or provision service resources for the VM.
preRelease(InstancePreReleaseRequest request)	Indicates the VM for a connected Instance is going to be destroyed
postRelease(InstancePostReleaseRequest request)	Indicates the destruction of the VM for a connected Instance is completed ***Note:*** The service adapter has the option to deregister/cleanup resources associated with the destroyed VM.
preReconfigure(InstancePreReconfigureRequest request)	Indicates the Template for a connected Instance is preparing to change a resource
postReconfigure(InstancePostReconfigureRequest request)	Indicates the resource change on a Template for a connected Instance is completed

InstanceRequest Class

The Services SDK `InstanceRequest` class is the basis for the parameter to the IInstanceLifecycle methods. `InstanceRequest` extends `ServiceProviderRequest` with the following methods:

Method	Type	Description
getInstance()	Instance	Returns the VM instance
getTemplate()	Template	Returns the template for the instance
getServiceInstance()	ServiceInstance	Returns the service instance asset connected to the VM instance

Method	Type	Description
getPeerServiceInstances()	List<ServiceInstance>	Returns a List of service instances that are the destination of connections to the template associated with the instance
getSrcConnections()	List<Connection>	Returns a List of all connections from the service instance to another Agility Platform asset
getDestConnections()	List<Connection>	Returns a List of all connections to the service instance from another Agility Platform asset
getDependencies()	List<Asset>	Returns a List of all Agility Platform assets upon which this service has a dependency
getDependents()	List<Asset>	Returns a List of all Agility Platform assets dependent upon this service

InstanceResponse Class

The Services SDK `InstanceResponse` class is wrapped by the promise returned by the `InstanceLifecycle` methods. `InstanceResponse` extends `ServiceProviderResponse` with the following methods:

Method	Type	Description
getInstance()	Instance	Returns the modified instance The value is null if not set.
setInstance(Instance value)	void	Used by the service adapter to set the modified instance from the request that triggered the response The Services framework subsequently persists the instance asset. The service adapter can skip setting the instance in the response if it does not modify the instance from the request.

InstanceOperations Methods

All of the `IInstanceLifecycle` methods use Services SDK derivations of `InstanceRequest` that provide no additional attributes:

■ `InstancePostBootRequest`

■ `InstancePostProvisionRequest`

- `InstancePostReconfigureRequest`

- `InstancePostReleaseRequest`

- `InstancePostRestartRequest`

- `InstancePostStartRequest`

- `InstancePostStopRequest`

- `InstancePreBootRequest`

- `InstancePreProvisionRequest`

- `InstancePreReconfigureRequest`

- `InstancePreReleaseRequest`

- `InstancePreRestartRequest`

- `InstancePreStartRequest`

- `InstancePreStopRequest`

Method invocations from workflows

`IInstanceLifecycle` method invocations can occur during the following instance-related Business Process Model and Notation (BPMN) workflows:

Workflow	Workflow node	IInstance method
instance-provision	PreProvisionCallout	preProvision(InstancePreProvisionRequest)
instance-provision	Provision Instance	preBoot(InstancePreBootRequest)
instance-provision	PostProvisionBefore	postProvision(InstancePostProvisionRequest)
instance-stop	PreStopCallout	preStop(InstancePreStopRequest)
instance-stop	PostStopCallout	postStop(InstancePostStopRequest)
instance-start	PreStartCallout	preStart(InstancePreStartRequest)
instance-start	PostStartBeforeCallout	postStart(InstancePostStartRequest)
instance-release	PreReleaseCallout	preRelease(InstancePreReleaseRequest)
instance-release	Release Instance	postRelease(InstancePostReleaseRequest)
instance-release	PostReleaseCallout	postRelease(InstancePostReleaseRequest)

> **Note:** The instance-restart workflow invokes the instance-stop and then instance-start workflow, resulting in a combination of the four method invocations.

Extending InstanceOperations

The Services SDK provides
`com.servicemesh.agility.sdk.service.operations.InstanceOperations` as a
default implementation of `IInstanceLifecycle`, for which every method simply returns a
completed promise.

A service adapter should extend `InstanceOperations` and override methods as needed. The
following is an example of a service adapter using the IP address from a recently provisioned
instance. This example uses AsyncLock to prevent simultaneous updates to the service in a
distributed Agility Platform environment.

```java
import com.servicemesh.agility.distributed.sync.AsyncLock;
@Override
public Promise<InstanceResponse> postProvision(InstancePostProvisionRequest
request)
{
    final Instance instance = request.getInstance();
    if (instance == null)
        return Promise.pure(new Exception("Instance not provided"));

    final ServiceInstance svcInstance = request.getServiceInstance();
    if (svcInstance == null)
        return Promise.pure(new Exception("Service Instance not provided"));

    final String svcName = PropHelper.getAssetPropertyAsString(
            "Name", svcInstance.getAssetProperties());

    // Lock at the service name so multiple requests don't modify the
    // service at the same time
    Promise<AsyncLock> lock = AsyncLock.lock("/agility/example-service/" +
            svcName + "/lock");

    return lock.flatMap(
        new Function<AsyncLock, Promise<InstanceResponse>>() {
            @Override
            public Promise<InstanceResponse> invoke(final AsyncLock alock)
            {
                String ip = instance.getPublicAddress();
                try {
                    IpUtil.ping(ip);
                }
```

```
                catch (Exception e) {
                    alock.unlock();
                    return Promise.pure(degradeInstance(instance,
                        "Instance is not reachable. "));
                }
                Promise<IHttpResponse> promise = addIpToService(svcInstance, ip);
                Promise<InstanceResponse> iPromise = promise.flatMap(
                    new Function<IHttpResponse, Promise<InstanceResponse>>() {
                        @Override
                        public Promise<InstanceResponse> invoke(IHttpResponse arg)
                        {

InstanceResponse response;

if (arg.getStatusCode() == 200) {

  // Set an asset property on the instance

  setAssetProperty("example-service", svcName,

    instance.getProperties());

  response = new InstanceResponse();

  response.getModified().add(instance);

  response.setStatus(Status.COMPLETE);

                        }
                        else {

response = degradeInstance(instance, "Unable to add IP to service");

                        }
                        return Promise.pure(response);
                    }
                });
            iPromise.onFailure(new Callback<Throwable>() {
                @Override
                public void invoke(Throwable t)
                {
                    alock.unlock();
                }
            });
            iPromise.onComplete(new Callback<InstanceResponse>() {
                @Override
                public void invoke(InstanceResponse t)
                {
```

```
                    alock.unlock();
                }
        });
        iPromise.onCancel(new Callback<Void>() {
            public void invoke(Void t)
            {
                alock.unlock();
            }
        });
        return iPromise;
    }});
}
private InstanceResponse degradeInstance(Instance instance, String degradeReason)
{
    InstanceResponse response = new InstanceResponse();
    response.setStatus(Status.FAILURE);
    response.setMessage(degradeReason);
    return response;
}
```

Service Instance (IServiceInstance)

The Agility Platform creates a ServiceInstance asset upon successful deployment of a Blueprint containing a Service.

The `com.servicemesh.agility.sdk.service.spi.IServiceInstance` interface returned by a service adapter `getServiceInstanceOperations()` represents the operations for a `ServiceInstance`.

These methods operate within the `Promise<ServiceProviderResponse>` type:

Method	Description
validate(ServiceInstanceValidateRequest request)	Validates the configuration of the service instance prior to deployment, update, or delete and prevents the event for an invalid configuration
provision(ServiceInstanceProvisionRequest request)	Provisions resources in the cloud upon service instance creation

Method	Description
reconfigure (ServiceInstanceReconfigureRequest request)	Revises any affected resources in the cloud after update of a service instance
start(ServiceInstanceStartRequest request)	Restarts a service from a stopped/suspended state
stop(ServiceInstanceStopRequest request)	Puts a running service into a stopped/suspended state
release(ServiceInstanceReleaseRequest request)	Releases any resources in the cloud associated with the service instance

ServiceInstanceRequest Class

The Services SDK `ServiceInstanceRequest` class is the basis for the parameter to the `IServiceInstanceLifecycle` methods. `ServiceInstanceRequest` extends `ServiceProviderRequest` with the following methods:

Method	Type	Description
getServiceInstance()	ServiceInstance	Returns the service instance
getOriginalServiceInstance()	ServiceInstance	During an update, returns the service instance as currently persisted ***Note:*** This method is useful for providing a comparison with attributes from the getServiceInstance() object.
getDependentServiceInstance()	ServiceInstance	During validation of an update to the service instance, returns a service instance asset that is connected to this service instance
getSrcConnections()	List<Connection>	Returns a List of all connections from this service instance to another Agility Platform asset
getDestConnections()	List<Connection>	Returns a List of all connections to this service instance from another Agility Platform asset
getDependencies()	List<Asset>	Returns a List of all Agility Platform assets upon which this service instance has a dependency
getDependents()	List<Asset>	Returns a List of all Agility Platform assets that are dependent upon this service

ServiceInstanceOperations Methods

Most of the `IServiceInstance` methods use Services SDK derivations of `ServiceInstanceRequest` that provide no additional attributes:

- `ServiceInstanceProvisionRequest`

- `ServiceInstanceReconfigureRequest`

- `ServiceInstanceReleaseRequest`

- `ServiceInstanceStartRequest`

- `ServiceInstanceStopRequest`

`ServiceInstanceValidateRequest` extends `ServiceInstanceRequest` with an additional method:

Method	Type	Description
getMode()	ValidateMode	Returns the type of operation performed: CREATE, UPDATE, or DELETE

During the update of a service instance, the service adapter always processes one `validate()` and one `reconfigure()` method where `getDependentServiceInstance()` for the request object returns null. This request focuses only on the service instance that is updated. However, the service adapter will process one additional `validate()` and `reconfigure()` method for each service instance upon which this service instance has a dependency—each design connection in the blueprint where this service instance is the source and the object returned by `getDependentServiceInstance()` is the destination. Each of these additional requests allow the service adapter to focus on the update in terms of a specific dependency relationship.

Method invocations from workflows

`IServiceInstance` method invocations can occur during the following service-related Business Process Model and Notation (BPMN) workflows:

Workflow	Workflow node	IServiceInstanceLifecycle methods
service-provision	SetStateStarting	validate(ServiceInstanceValidateRequest) reconfigure(ServiceInstanceReconfigureRequest) provision(ServiceInstanceProvisionRequest)
service-stop	SetStateStopping	validate(ServiceInstanceValidateRequest) reconfigure(ServiceInstanceReconfigureRequest) stop(ServiceInstanceStopRequest)
service-start	SetStateStarting	validate(ServiceInstanceValidateRequest) reconfigure(ServiceInstanceReconfigureRequest) start(ServiceInstanceStartRequest)
service-restart	SetStateStopping	validate(ServiceInstanceValidateRequest) reconfigure(ServiceInstanceReconfigureRequest) stop(ServiceInstanceStopRequest) start(ServiceInstanceStartRequest)
service-release	SetStateStopping	validate(ServiceInstanceValidateRequest) reconfigure(ServiceInstanceReconfigureRequest) release(ServiceInstanceReleaseRequest)
service-delete	PreDelete Callout	validate(ServiceInstanceValidateRequest)

Extending ServiceInstanceOperations

The Services SDK provides `com.servicemesh.agility.sdk.service.operations.ServiceInstanceOperations` as a default implementation of `IServiceInstance` for which every method simply returns a completed promise.

A service adapter should extend `ServiceInstanceOperations` and override methods as needed. The following is an example of a validation that is invoked when a service instance is created, updated, or deleted:

```
@Override
public Promise<ServiceProviderResponse> validate(ServiceInstanceValidateRequest
request)
{
```

```
    ServiceInstance updated = request.getServiceInstance();
    if (updated == null)
        return Promise.pure(new Exception("Service Instance not provided."));

    if (request.getDependentServiceInstance() != null)
        return doValidateDependency(request);

    ValidateMode mode = request.getMode();
    if (mode != null) {
        if (mode == ValidateMode.CREATE) {
            return doValidate(request);
        }
        else if (mode == ValidateMode.UPDATE) {
            ServiceInstance original = request.getOriginalServiceInstance();
            if (original == null)
                return Promise.pure(new Exception("Original Service Instance not
                provided."));

            if (original.getState() != ServiceState.UNPROVISIONED) {
                String originalName =
                    PropHelper.getAssetPropertyAsString(
                        "Name", original.getAssetProperties());
                String updatedName =
                    PropHelper.getAssetPropertyAsString(
                        "Name", updated.getAssetProperties());

                if (! originalName.equals(updatedName))
                    return Promise.pure(new Exception("Provisioned Service Instance
                    Name cannot be changed."));
            }
            else {
                // Perform regular validation when updating an unprovisioned
                service instance
                return doValidate(request);
            }
        }
    }
    return super.validate(request);
}
```

Service Instance Lifecycle (IServiceInstanceLifecycle)

Just as a Blueprint can contain design connections between Workloads and Services, it can also contain design connections between two Services. The Service-Service design connection also has a Source end and a Destination end, but the source ServiceInstance asset is dependent upon the destination ServiceInstance asset.

The Service SDK `IServiceInstanceLifecycle` interface represents the operations for a service adapter to consider when a ServiceInstance bound to another ServiceInstance changes state.

These methods operate within the `Promise<ServiceProviderResponse>` type:

Method	Description
preProvision(ServiceInstancePreProvisionRequest request)	Indicates a connected ServiceInstance is beginning its provisioning workflow ***Note:*** The service adapter has the option to abort the provisioning if it cannot support another connected service.
postProvision(ServiceInstancePostProvisionRequest request)	Indicates the service provisioning in the cloud for a connected ServiceInstance is completed ***Note:*** The service adapter has the option to register the service with a service offering and/or allocate resources for the service.
preStop(ServiceInstancePreStopRequest request)	Indicates the service for a connected ServiceInstance is preparing to stop
postStop(ServiceInstancePostStopRequest request)	Indicates the service for a connected ServiceInstance is stopped without releasing its underlying resources ***Note:*** The service adapter has the option to deregister/cleanup resources associated with the stopped service.
preStart(ServiceInstancePreStartRequest request)	Indicates the service for a connected ServiceInstance that is in a stopped state is preparing to start ***Note:*** The service adapter has the option to verfify that the service settings are compatible with the service.

Method	Description
postStart(ServiceInstancePostStartRequest request)	Indicates the start of a service for a connected ServiceInstance is completed **Note:** The service adapter has the option to register or provision service resources for the service.
preRestart(ServiceInstancePreRestartRequest request)	Indicates the service for a connected ServiceInstance is preparing to restart
postRestart(ServiceInstancePostRestartRequest request)	Indicates the restart of the service for a connected ServiceInstance is completed **Note:** The service adapter has the option to register or provision service resources for the service.
preRelease(ServiceInstancePreReleaseRequest request)	Indicates the service for a connected ServiceInstance is going to be destroyed
postRelease(ServiceInstancePostReleaseRequest request)	Indicates the destruction of the service for a connected ServiceInstance is completed **Note:** The service adapter has the option to deregister/cleanup resources associated with the destroyed service.

ServiceInstanceLifecycleRequest Class

The Services SDK `ServiceInstanceLifecycleRequest` class is the basis for the parameter to the `IServiceInstanceLifecycle` methods. `ServiceInstanceLifecycleRequest` extends `ServiceInstanceRequest` with the following method:

Method	Type	Description
getPeerServiceInstances()	List<ServiceInstance>	Returns a List of service instances that are the destination of connections to the cycling service instance

ServiceInstanceLifecycleOperations Methods

All of the `IServiceInstanceLifecycle` methods use a Services SDK derivation of `ServiceInstanceRequest` that provides no additional attributes:

■ `ServiceInstancePostProvisionRequest`

- `ServiceInstancePostReleaseRequest`

- `ServiceInstancePostRestartRequest`

- `ServiceInstancePostStartRequest`

- `ServiceInstancePostStopRequest`

- `ServiceInstancePreProvisionRequest`

- `ServiceInstancePreReleaseRequest`

- `ServiceInstancePreRestartRequest`

- `ServiceInstancePreStartRequest`

- `ServiceInstancePreStopRequest`

Method invocations from workflows

`IServiceInstanceLifecycle` method invocations can occur during the following service-related Business Process Model and Notation (BPMN) workflows:

Workflow	Workflow node	IServiceInstance method
service-provision	PreProvisionCallout	preProvision(ServiceInstancePreProvisionRequest)
service-provision	PostProvisionCallout	postProvision(ServiceInstancePostProvisionRequest)
service-stop	PreStopCallout	preStop(ServiceInstancePreStopRequest)
service-stop	PostStopCallout	postStop(ServiceInstancePostStopRequest)
service-start	PreStartCallout	preStart(ServiceInstancePreStartRequest)
service-start	PostStartCallout	postStart(ServiceInstancePostStartRequest)
service-release	PreReleaseCallout	preRelease(ServiceInstancePreReleaseRequest)
service-release	PostReleaseCallout	postRelease(ServiceInstancePostReleaseRequest)

> *Note:* The service-restart workflow invokes the service-stop and then the service-start workflow , resulting in a combination of the four method invocations.

Extending ServiceInstanceLifecycleOperations

The Services SDK provides `com.servicemesh.agility.sdk.service.operations.ServiceInstanceLifecycleOperations` as a default implementation of `IServiceInstanceLifecycle`, for which every method simply returns a completed promise.

A service adapter should extend `ServiceInstanceLifecycleOperations` and override methods as needed. In the following example, a service instance is released and then removed from the service supported by the adapter.

```
@Override
public Promise<ServiceProviderResponse> preRelease(final
ServiceInstancePreReleaseRequest request)
{
    final ServiceInstance releasingService = request.getDependentServiceInstance
    ();
    if (releasingService == null)
        return Promise.pure(new Exception("Dependent Service Instance not
        provided"));
    final ServiceInstance myService = request.getServiceInstance();
    if (myService == null)
        return Promise.pure(new Exception("Service Instance not provided"));

    Promise<IHttpResponse> promise = dropReleasedService(myService,
    releasingService);
    return promise.flatMap(
        new Function<IHttpResponse, Promise<ServiceProviderResponse>>() {
            @Override
            public Promise<ServiceProviderResponse> invoke(IHttpResponse arg)
            {
                ServiceProviderResponse response = new ServiceProviderResponse();
                if (arg.getStatusCode() == 200) {
                    rmAssetProperty("example-service",
                            releasingService.getAssetProperties());
                    response.getModified().add(releasingService);
                }
                else {
                    // We don't need to degrade a releasing service instance.
                    // Just log a message
                    MyServiceLifecycleOperations.logger.info("dropReleaseService()
                    - " +
                        myService.getName() + " dropping " +
                        releasingService.getName() + " returned non-OK status");
                }
                response.setStatus(Status.COMPLETE);
                return Promise.pure(response);
            }
        });
```

```
}
```

Asset Lifecycle (IAssetLifecycle)

The AssetOperations exist to allow service adapters to receive notifications and perform any necessary actions when a CRUD operation is executed for an associated Asset.

The Service SDK `IAssetLifecycle` interface represents the operations for a service adapter to consider when an Asset associated with the service changes state.

These methods operate within the `Promise<ServiceProviderResponse>` type:

Method	Description
preCreate(PreCreateRequest request)	Used by the service adapter to validate the Asset
postCreate(PostCreateRequest request)	Called after a successful creation of the Asset ***Note:*** This is useful if a service adapter should perform some action based on the newly created Asset.
preUpdate(PreUpdateRequest request)	Used by the service adapter to validate the Asset prior to the Asset update
postUpdate(PostUpdateRequest request)	Called after a successful update of the Asset ***Note:*** This is useful if a service adapter should perform some action based on the updated Asset.
preDelete(PreDeleteRequest request)	Used by the service adapter to validate the system is in a valid state to delete the Asset
postDelete(PostDeleteRequest request)	Called after a successful deletion of an Asset ***Note:*** This is useful if a service adapter should unregister some information resulting from the removal of the Asset.

AssetOperations Request Classes

To support the AssetOperations for a service adapter, there are six Request classes:

- `PreCreateRequest`
- `PostCreateRequest`
- `PreUpdateRequest`
- `PostUpdateRequest`

- `PreDeleteRequest`

- `PostDeleteRequest`

All six of these classes extend `ServiceInstanceRequest` with the following methods:

Method	Type	Description
getAsset()	Asset	Returns the asset that is changing state
setAsset(Asset value)	void	Used by the Services framework to populate the Asset in the request

Registration of Asset

In order for the service adapter to use AssetOperations, it must know the type of Asset(s) on which to perform lifecycle operations. This is done by adding the Asset class name to the `AssetLifecycles` list in the `RegistrationRequest` that is returned in `getRegistrationRequest`. For more information about the `getRegistrationRequest` method, see .

Extending Asset Operations

The Services SDK provides `com.servicemesh.agility.sdk.service.operations.AssetOperations` as a default implementation of `IAssetLifecycle`, for which every method simply returns a completed promise.

> **Important:** The service adapter has no way to persist any changes made to the Asset or the Service Instance.
>
> Returning a failed promise does not abort the Asset CRUD operation.

A service adapter should extend `AssetOperations` and override methods as needed. The following is an example of a service adapter updating the list of address ranges to reflect a change made to a Network Asset.

First, you need to register the Network class so that lifecycle events for Network Assets will trigger the overridden AssetOperations methods:

```
public class MyAdapter extends ServiceAdapter
{
    @Override
    public RegistrationRequest getRegistrationRequest()
```

```
    {
            RegistrationRequest registration = new RegistrationRequest();
            // set up various service properties...
            // ...
            // ...
            // register the Network AssetLifecycle for our AssetOperations methods.
            registration.getAssetLifecycles().add
            ("com.servicemesh.agility.api.Network");
            return registration;
    }
}
```

Then you extend the AssetOperations and override the desired method:

```
public class MyAssetOperations extends AssetOperations
{
    @Override
    public Promise<ServiceProviderResponse> postUpdate(PostUpdateRequest request)
    {
        final Network network = (Network) request.getAsset();
        Promise<ServiceProviderResponse> promise = updateAddressRanges(network);
        return promise;
    }


    private Promise<ServiceProviderResponse> updateAddressRanges(Network network)
    {
        // Method to update the address ranges for a Network in an external
        service
    }
}
```

Connection (IConnection)

The Agility Platform REST API provides methods for affecting deployed connections between VM templates and service instances (or between service instances). Invocation of a POST, PUT, or DELETE method triggers the service adapter connection operations if either end of the connection has a service provider associated with the adapter.

Action	REST type	URI
Delete a connection	DELETE	connection/{id}
Get all connections	GET	connection/
Search for connections	GET	connection/search
Get connection by ID	GET	connection/{id}
Create a connection	POST	connection/
Update a connection	PUT	connection/{id}

The Service SDK `IConnection` interface represents the operations for a service adapter to consider when a Connection is created, updated, or deleted.

These methods operate within the `Promise<ServiceProviderResponse>` type:

Method	Description
preCreate(ConnectionPreCreateRequest request)	Used by the adapter to validate the configuration of the connection prior to creation and allow/reject the settings
postCreate(ConnectionPostCreateRequest request)	Called after successful creation of the connection *Note:* This is useful if a service adapter should push this configuration to the actual service provider.
preUpdate(ConnectionPreUpdateRequest request)	Used by the adapter to validate the configuration of the connection prior to an update and allow/reject the settings
postUpdate(ConnectionPostUpdateRequest request)	Called after successful update of the connection *Note:* This is useful if a service adapter should push this configuration to the actual service provider.

Method	Description
preDelete(ConnectionPreDeleteRequest request)	Used by the adapter to validate the system is in a valid state to perform a delete
postDelete(ConnectionPostDeleteRequest request)	Called after successful deletion of the connection ***Note:*** This is useful if a service adapter should clean up any actual service provider resources associated with the connection.

ConnectionRequest Class

The Services SDK `ConnectionRequest` class is the basis for the parameter to the `IConnection` methods. `ConnectionRequest` extends `ServiceProviderRequest` with the following method:

Method	Type	Description
getConnection()	Connection	Returns the Connection asset
getSource()	Asset	Returns the asset at the source end of the connection
getDestination()	Asset	Returns the asset at the destination end of the connection
getDestConnections()	List<Connection>	Returns a List of all connections to the service instance
getDepenents()	List<Connection>	Returns a List of assets dependent on this service instance

ConnectionOperations Methods

All of the `IConnection` methods use a Services SDK derivation of `ConnectionRequest` that provides no additional attributes:

- `ConnectionPostCreateRequest`
- `ConnectionPostDeleteRequest`
- `ConnectionPostUpdateRequest`
- `ConnectionPreCreateRequest`
- `ConnectionPreDeleteRequest`

■   `ConnectionPreUpdateRequest`

The Services SDK provides
`com.servicemesh.agility.sdk.service.operations.ConnectionOperations` as a
default implementation of `IConnection`, for which every method simply returns a completed
promise.

A service adapter should extend `ConnectionOperations` and override methods as needed.
In the following example, the same validation is performed when a connection is created or
updated.

```
@Override
public Promise<ServiceProviderResponse> preCreate(ConnectionPreCreateRequest
request)
{
    return doValidate(request);
}
@Override
public Promise<ServiceProviderResponse> preUpdate(ConnectionPreUpdateRequest
request)
{
    return doValidate(request);
}
private Promise<ServiceProviderResponse> doValidate(ConnectionRequest request)
{
    if ((request.getDestination() != null) &&
        (request.getDestination() instanceof ServiceInstance)) {
        ServiceInstance dest = (ServiceInstance)request.getDestination();

        // This service adapter allows its service to be either end of
        // a connection. Verify that the destination of this connection
        // is this adapter's service before validating.
        if (isMyService(dest.getAssetType())) {
            try {
                // Verify that a location is defined for the source
                verifyLocation(request.getSource(), request.getClouds(),
                        request.getServiceProviders());
            }
            catch (Exception ex) {
                return Promise.pure(ex);
            }
        }
```

```
    }

ServiceProviderResponse response = new ServiceProviderResponse();

response.setStatus(Status.COMPLETE);

return Promise.pure(response);

}
```