Getting Started with the webMethods Application Platform API

Version 9.10

April 2016

Document ID: AP-API-GS-910-20160415

Table of Contents

About this Guide

This guide describes webMethods Application Platform API services. It provides reference information for developers who want to build additional functionality on top of their Application Platform projects.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Introduction to the Application Platform API

About Application Platform API

webMethods Application Platform API enables you to build additional functionality to your Application Platform projects. You can use the Application Platform API to execute the following tasks:

- Publish plain old Java objects (POJOs) as OSGi Services.
- Inject service dependencies into other services.
- Look up services from the OSGi registry.
- Expose POJO classes as Integration Server assets.
- Generate tests with the Application Platform integration test framework.
- Enable authentication and authorization by adding single sign-on functionality.

2 Using the Application Platform API

Publishing POJOs as OSGi Services

Use the following annotations to publish POJOs as OSGi services.

@Service

Use this annotation to mark a POJO class to be exposed as an OSGi service. Specify @Service on the class type.

For example:

```
@Service(name = "my-service", init = "start", destroy = "stop", ranking = "10",
interfaces = { "com.example.MyInterface" }, properties = { @Property(key =
"key1", values = {1, 2, 3}, valueType = "java.lang.Integer") })
public class MyService implements MyInterface {
}

interface MyInterface {
}
```

The following table lists the properties of @Service:

Property	Default Value	Description
name	Simple name of the annotated class	**String** Optional. The name of the bean backing this service. If you do not specify a value, this property defaults to the simple name of the bean class.
value	Simple name of the annotated class	**String** Optional. An alternative way to specify the name of the service bean. This property is useful when you do not specify any other attributes.
ranking	0	**Integer** Optional. The ranking value to be published as the service.ranking property for this service to distinguish.
init	""	**String** Optional. The method to invoke when the bean that backs the service is initialized.
destroy	""	**String** Optional. The method to invoke when the bean that backs the service is destroyed.
interfaces	The fully qualified name (FQN) of the annotated class	**String List** Optional. The list of interfaces, under which the service will be published. If you do not specify a value for this property, the service

Property	Default Value	Description
		will only be published under the name of the implementation class.
dependsOn	`""`	**String** Optional. Used to express a dependency on another component that must be fully initialized before this service can be initialized and exported.
properties	`{}`	**String** Optional. The list of service properties to be published with the service.

@Property

Use this annotation to declare the properties for the service. You can add more than one value for the key. Optionally, you can also specify the type of the key and the type of the values.

The following table lists the properties of @Property:

Property	Default Value	Description
key	`""`	**String** Required. The name or key of the property.
values	`{}`	**Sting List** Required. The values to be associated with the property name.
valueType	`java.lang.Sring`	**String** Optional. The type of the values of this property.

The following example shows the GreeterImpl POJO class registered as an OSGi service under the name "greeter-impl", as well as two interfaces and one service property.

```
public interface IGreeter {
       public String greetMe(String name);
}

@Service(
                          name="greeter-impl",
                          interfaces = {"com.example.osgi.greet.api.IGreeter",
"org.osgi.service.cm.ManagedService"},
                          properties = {@Property(key="service.pid", val-
ues="com.example.osgi.greet")}
                                )
public class GreeterImpl implements IGreeter, ManagedService {
            @Override
       public String greetMe(String name) {
              return "Hello, " + name;
}
}
```

Injecting Service Dependencies into Other Services

Use the following annotation to inject service dependencies into other services.

@ServiceReference

Use this annotation to inject a service from the runtime registry into another service being published (using the @Service annotation). This provides a form of dependency injection, in which the injected dependency is another POJO/bean already published in the runtime as an OSGi service.

You must specify a setter method to set the injected POJO reference in the same class that accompanies the field declaration. This is the class that contains the @ServiceReference annotation.

The following table lists the properties of @ServiceReference:

Property	Default Value	Description
id	`""`	**String** Required. A unique identifier for this service reference. The specified id must not be in conflict with any other implicit or explicit @Service annotation name attribute value.
interfaces	`{}`	**String List** Required if the filter property is not specified, otherwise it is optional. The interfaces that the service reference proxy should implement when it is wired in from the service registry. A service that implements these interfaces must be available in the registry. At least one interface or class name must be specified for this service reference.
filter	`""`	**String** Required if the interfaces property is not specified, otherwise it is optional. An OSGi filter expression that constrains the service registry lookup to only those services that match the given filter. The filter string is in the following format: (property-name = value). For example, (asynchronous-delivery=true) restricts the service lookup to those services that have a property with name asynchronous-delivery that is set to `true`.
timeout	`5000` ms	**Integer** Optional. The amount of time (in milliseconds) to wait for a backing service to

Property	Default Value	Description
		become available when an operation is invoked. If no matching service becomes available within this timeout period, an unchecked ServiceUnavailableException is thrown.
componentName	`""`	**String** Optional. A convenient shortcut for specifying a filter expression that matches the property named org.eclipse.gemini.blueprint.bean.name that is automatically advertised for beans, published with the @Service annotation.
dependsOn	`""`	**String** Optional. Specifies that the service reference should not be looked up in the service registry until the named dependent bean has been instantiated.
availability	`Availability.OPTIONAL`	**ServiceReference.Availability** Optional. Indicates the requirement for the availability of this service reference. By default, the reference is treated as an optional requirement. If set to `MANDATORY`, then the @Service registration will only succeed if the referenced service is already available. **Important:** Do not declare a mandatory reference to a service that is also exported by the same bundle. This can cause application context creation to fail through either deadlock or timeout.

The following example shows the GreeterImpl class published as an OSGi service that depends on the ResourceUtil class that is in turn published as another OSGi service.

```
@Service(name = “greeter-impl”, interfaces =
{ “com.example.osgi.greet.api.IGreeter”,
                      “org.osgi.service.cm.ManagedService” }, properties =
{ @Property(key = “service.pid”, values = “com.example.osgi.greet”) })


public class GreeterImpl implements IGreeter, ManagedService {
              public static final String KEY_HELLO = “hello”;
              private String key = KEY_HELLO;

             @ServiceReference(id = “resourceUtilRef”, interfaces =
{“com.example.osgi.greet.impl.ResourceUtil”})
                ResourceUtil resUtil;

                public void setResUtil(ResourceUtil resUtil) {
                             this.resUtil = resUtil;
                }

                ...
}

@Service
public class ResourceUtil {
             ...
}
```

Looking up Services from the OSGi Registry

Class and Description
com.softwareag.applatform.sdk.ServiceUtil A helper class that provides utility methods when working with OSGi services. Use this class to look up registered services.

Public API Methods in ServiceUtil Class

The following table lists the public API methods in ServiceUtil class:

Method Name	Return Type	Method Arguments	Description
getService	`T`	`ServletContext servletCtxClass<T> serviceCls`	Returns the instance of the OSGi service of type serviceCls from the specified ServletContext. This method looks for an instance of BundleContext in the ServletContext under the attribute name osgi-bundlecontext and use the obtained BundleContext to look up the service.
getService	`T`	`Class<T> serviceClsBundleContext bundleCtx`	Gets the OSGi service of given serviceCls type using the given BundleContext. If no service of the

Method Name	Return Type	Method Arguments	Description
			serviceCls type is registered, this method returns a null value.
getBundleContext	BundleContext	`Class<?> bundleCls`	Gets the BundleContext from the bundle containing the given class. If there is no BundleContext specified, this method returns a null value.
getService	`T`	`Class<T> serviceCls`	Gets the OSGi service for the given service class type. If no service of the specified type is registered, this method returns a null value.

Configuring POJO Services Dynamically

Application Platform enables you to dynamically configure a published POJO service by using the @Service annotation. For more information about the @Service annotation, see "Publishing POJOs as OSGi Services" on page 10.

For information about how to enable dynamic service configuration in Application Platform projects, see *webMethods Application Platform User's Guide*.

The following table outlines the related API documentation:

Class and Description
org.osgi.service.cm.ManagedService For information, see the OSGi documentation.

The following methods must be implemented from the ManagedService interface:

Method Name	Return Type	Method Arguments	Description
update	`void`	`java.util.Dictionary<java.lang.String,?> properties`	For information about the updated method, see the OSGi documentation.

Exposing POJO classes as Integration Server Assets

This section describes the annotations you can use for exposing POJO classes as Integration Server assets.

@ExposeToIS

This annotation is used to identify a class that contains one or more methods to be exposed as Integration Server services. It is combined with the @Service and @ExposedMethod annotations to support the presentation of methods in a Java POJO as IS services. Since the generated Integration Server assets assume that the Java class is registered in OSGi as a service, this annotation must be used with the @Service annotation.

For example:

```
@ExposeToIS(packageName="OrdersService")
public class OrdersServiceImpl implements OrdersService {
}
```

The following table lists the properties of @ExposeToIS:

Property	Default Value	Description
packageName	`""`	**String** Optional. The name of the Integration Server package where services from this class are created. Note that this is the name of an Integration Server package, not a Java package. If no value is provided, when the Integration Server service is generated, the value of the @Service.name property will be used as the Integration Server package name.

@ExposedMethod

This annotation identifies a method to be exposed as an Integration Server service. It is valid only on public methods. Since Integration Server does not support service name overloading, there are restrictions on exposing methods from a Java class. If the exposed Java class defines methods using overloaded names, only one method with a given name can be exposed.

This annotation has no properties.

For example:

```
@ExposedMethod
public String createReceipt(Order inOrder) {
}
```

Example of Using the @ExposeToIS and the @ExposedMethod Annotations

In the following example the OrdersServiceImpl class implements the OrdersService interface, which declares several methods, including @ExposeToIS and @ExposeToIS. When this POJO is published in an Application Platform project, several artifacts are created in the Integration Server namespace.

As a result of the packageName property, an Integration Server package, named OrdersService is created, if necessary. Based on the name of the Java package, where the OrdersService interface is defined, a folder, named 'com.softwareag.demp.orders.api', is created. This folder is located in the new Integration Server package.

Each of the exposed methods creates an Integration Server service in the new folder. The service name matches the exposed method name. The signatures for these new IS services match the method signatures. For example, the orderReceipt service signature includes a String output and one input of type Document, named inItem, where the document structure matches the properties of the Order POJO.

```
package com.softwareag.demo.orders.impl;

@Service(name="RegisteredOrdersService", interfac-
es={"com.softwareag.demo.orders.api.OrdersService"})
@ExposeToIS(packageName="OrdersService")
public class OrdersServiceImpl implements OrdersService {

      @Override
      @ExposedMethod
      public float calculateCharge(LineItem inItem) {
             ....
      }

      @Override
      @ExposedMethod
      public String createReceipt(Order inOrder) {
...
      }
  }

public interface OrdersService {
      public String createReceipt(Order inOrder);
      public float calculateCharge(LineItem inItem);
      ...
}
```

If the packageName property is omitted from this example code, the package in the Integration Server namespace will be named RegisteredOrdersService, based on the @Service annotation.

Generating Tests with the Application Platform Integration Test Framework

This section describes the main classes and annotations that you should use when you develop JUnit tests in the Application Platform integration test framework. The classes are available in the Application Platform API Libraries classpath container.

For more information about the Application Platform API Libraries container, see *webMethods Application Platform User's Guide*.

For more information about JUnit testing, including classes and annotations, see the JUnit website at http://junit.org.

Class and Description
com.softwareag.applatform.sdk.test.framework.AppPlatformIntegrationTest An abstract base class, from which your test classes can inherit in order to use the JUnit runner.
com.softwareag.applatform.sdk.test.framework.IntegrationTestRunner The main class that drives the Application Platform integration test framework.
com.softwareag.applatform.sdk.test.framework.AppPlatformIntegrationTestWithParameters A convenience abstract base class that your test classes can inherit from when you create parameterized tests.
com.softwareag.applatform.sdk.test.framework.ParameterizedIntegrationTestRunner A custom JUnit runner that supports running parameterized tests in the Application Platform integration test framework.

Annotation and Description
com.softwareag.applatform.sdk.annotations.TestBundle A required class-level annotation that must be specified on every test class that should be executed within the Application Platform integration test framework.

Annotation and Description
com.softwareag.applatform.sdk.annotations.RunOnServer Used to specify the details of the server, on which the bundle that hosts the test class exists and the test class is executed.

Non-Parameterized Tests

Use the following classes when you create non-parameterized tests.

AppPlatformIntegrationTest

The com.softwareag.applatform.sdk.test.framework.AppPlatformIntegrationTest class is an abstract base class, from which your test classes can inherit in order to use the JUnit runner.

This class provides no-op implementations for the following methods, which can be overridden:

Annotation Name	Method Name
@BeforeClass	setupClass
@AfterClass	destroyClass
@Before	setup
@After	Destroy

IntegrationTestRunner

The com.softwareag.applatform.sdk.test.framework.IntegrationTestRunner class is the main class that drives the Application Platform integration test framework. This class is a custom JUnit runner class and it is activated through the JUnit @RunWith annotation.

If you use the AppPlatformIntegrationTest class as the base class of your tests, you do not have to use the IntegrationTestRunner class directly in your tests. You need to use the IntegrationTestRunner class only if your test class already extends from another base class and it cannot extend from AppPlatformIntegrationTest.

The IntegrationTestRunner class performs the following key steps:

1. Validates that the test class contains the @TestBundle annotation with the bundle symbolic name and the bundle version, if it is specified.

2. Initiates a JMS client connection to the configured server by using the details from the @RunOnServer annotation, if specified. If the @RunOnServer annotation is not specified, the class uses the default values.
3. Verifies that the bundle that contains the test class is deployed and active on the running server.
4. Executes the annotated @Test methods in the test class by making a JMX call to the actual test class that is hosted in the project bundle.
5. Captures success and failure messages of the test run and reports them to the JUnit and the Console view in Designer.
6. Terminates the JMX client connection when the test class is executed.

Parameterized Tests

Use the following classes when you create parameterized tests.

AppPlatformIntegrationTestWithParameters

The com.softwareag.applatform.sdk.test.framework.AppPlatformIntegrationTestWithParameters class is a convenience abstract base class, from which your test classes can inherit when you create parameterized tests.

This class provides no-op implementations for the following methods, which can be overridden:

Annotation Name	Method Name
@BeforeClass	setupClass
@AfterClass	destroyClass
@Before	setup
@After	Destroy

ParameterizedIntegrationTestRunner

The com.softwareag.applatform.sdk.test.framework.ParameterizedIntegrationTestRunner class is a custom JUnit runner that supports running parameterized tests in the Application Platform integration test framework. This class is activated through the @RunWith annotation.

If you use the AppPlatformIntegrationTestWithParameters class as the base class of your tests, you do not have to use this class directly in your test class. You need to use the

ParameterizedIntegrationTestRunner class only if your test class already extends from another base class and it cannot extend from AppPlatformIntegrationTestWithParameters.

The ParameterizedIntegrationTestRunner class performs the following key steps:

1. Validates that the test class contains the @Parameters annotation on a method that provides the test data.
2. For each set of parameters in the test data, creates an instance of the other child runner that is responsible for running the test methods in the test class.
3. Sets the name of the test by using the name attribute of the @Parameters annotation.
4. Executes the child test runner.

Test Class Annotations

Use the following annotations for the test classes you create for your Application Platform integration tests.

@TestBundle

The @TestBundle annotation is a required class-level annotation. You must specify this annotation on every test class that should be executed within the Application Platform integration test framework.

The following table lists the properties of @TestBundle:

Property Name	Default Value	Description
symbolicName	The project name of the corresponding Application Platform project.	**String** Required. The symbolic name of the bundle that contains this test class when it is deployed to the configured server runtime. This value corresponds to the `Bundle-SymbolicName` OSGi header value.
version	*1.0.0*	**String** Optional. The version of the bundle that hosts the corresponding test class. This value corresponds to the `Bundle-Version` OSGi header value.

@RunOnServer

Use the @RunOnServer annotation to specify the details of the server, on which the bundle hosting the test class exists, and where the test class is executed. Do not use @RunOnServer if the configured server uses the same default values, as the default values of the annotation. The default values of @RunOnServer correspond to the default values of a local Integration Server instance. However, if any of the server properties are different from the default values, you must specify the @RunOnServer annotation at the test class level.

Note: If you are using My webMethods Server, note that its default JMX port value is different.

The following table lists the properties of @RunOnServer:

Property Name	Default Value	Description
host	*localhost*	**String** Optional. The host name of the server, on which the Application Platform project bundle is deployed.
jmxPort	*8075*	**Integer** Optional. The JMX port of the configured server.
username	*Administrator*	**String** Optional. The JMX client connection username.
pwd	*manage*	**String** Optional. The JMX client connection password.
timeout	*15000*	**Integer** Optional. The JMX client connection timeout value in milliseconds.

Examples of Using the Application Platform Integration Test Framework

This section provides examples of a non-parameterized and a parameterized test in the Application Platform integration test framework.

Example of a Non-Parameterized Test

The following example shows the GreeterImpl class published as an OSGi service that depends on the ResourceUtil class that is in turn published as another OSGi service.

In the example, the test class for the GreeterImpl class verifies that the IGreeter API implementation is correctly registered as an OSGi service and is accessible using the ServiceUtil class.

The @TestBundle annotation specifies the symbolic name of the project bundle that contains this test class.

The @RunOnServer annotation is explicitly specified. However, it is not required, as it uses the default values.

The test class inherits from the AppPlatformIntegrationTest class and it implicitly uses the IntegrationTestRunner JUnit custom runner.

```
@TestBundle(symbolicName="greeter-service")
@RunOnServer(host="localhost", jmxPort=8075, username="Administrator",
pwd="manage")
public class GreeterImplTest extends AppPlatformIntegrationTest {
    @Test
    public void testGreeterServiceRegistered() throws Exception {
        IGreeter greeter = ServiceUtil.getService(IGreeter.class);
        assertNotNull(greeter);

        String result = greeter.greetMe("test");
        assertNotNull(result);
        assertTrue(result.contains("test"));
        assertTrue(greeter instanceof GreeterImpl);
        System.out.println("Passed!");
    }
}
```

Example of a Parameterized Test

The following example shows a simple parameterized test that runs in the Application Platform integration test framework. The example consists of the following parts:

- A simple POJO class, named Hello. This class returns a greeting string for a provided input name.
- A JUnit test that tests the Hello class. The test uses the @Parameters annotation and the Application Platform parameterized integration test support.

```
public class Hello {
    String name;
    Hello(String name) {
        this.name = name;
    }
    String greet() {
        return String.format("Hello, %s", name == null ? "Guest!" : name);
    }
}
```

```
@TestBundle(symbolicName = "HelloBundle")
@RunOnServer(jmxPort = 8075)
public class HelloTest extends AppPlatformIntegrationTestWithParameters {
    @Parameters(name = "test-{index}-with-name-{0}")
    public static Iterable<Object[]> data() {
       return Arrays.asList(new Object[][] { { "abc", "Hello, abc" },
       { null, "Hello, Guest!" } });
    }

    @Parameter(0)
    public String name;
    @Parameter(1)
    public String greeting;

    @Test
    public void test() {
       String result = new Hello(name).greet();
       System.out.println("Got result: " + result + " for input name: " + name);
       assertEquals(greeting, result);
    }
}
```

Adding Single Sign-On Authentication to Application Platform Projects

Application Platform supports SSO authentication. To add SSO to your Application Platform projects, you can use the available security filter, class, or annotation, which are described here. The class and the annotation are available in the Application Platform API Libraries classpath container.

For more information about the Application Platform API Libraries container, see *webMethods Application Platform User's Guide*.

Web Application Layer Security

The following table describes the properties and values of the security filter that you add in the web.xml file.

For information about how to enable SSO in your Application Platform web projects by adding the security filter, see *webMethods Application Platform User's Guide*.

Filter and Description
com.softwareag.applatform.security.filter.AppPlatformSecurityFilter A servlet filter that is added to the web.xml file of the required Application Platform web project. Supports SSO functionality for web applications.

The following table lists the properties of AppPlatformSecurityFilter:

Property	Value	Description
realm	`AppPlatformRealm` This is the only valid value.	**String** Required. The Application Platform realm.
nextauthMethod	Valid values: ■ `BASIC` Basic authentication. ■ `CLIENT_CERT` Authentication with client certificate.	**String** Required. The next authentication method to try if the current authentication request fails. For detailed information about the possible values, see the Java EE documentation, provided by Oracle.

Property	Value	Description
	■ `FORM` Form authentication. Requires attributes for user name and the password, as follows: j_username j_password	
roleNamesAllowed	Depends on the runtime server, as follows: ■ For Integration Server the roles must be equivalent to the roles in the Integration Server access control list (ACL). For information about the ACL, see *webMethods Integration Server Administrator's Guide*. ■ For My webMethods Server the roles must be equivalent to the applicable Security Realm container. For information about Security Realm containers, see *Administering My webMethods Server*.	**String List** Optional. A comma-separated list of allowed user roles. Users are authenticated when they have one of the listed roles.
formLoginPage		**String** Optional. The address of the login page when using the form authentication type.
formErrorPage		**String** Optional. The address of the error page that displays when the form authentication fails.

OSGi Service Layer Security

The following tables describe the class and annotation that are provided by the Application Platform API for implementing security at the OSGi service layer. When you implement OSGi service layer security, you can add one of the following types of SSO to your application:

- Declarative security, in which the users that are allowed to have access to the application are determined statically.

  For more information about the declarative security mechanism, see "Declarative Security" on page 27.

- Dynamic runtime security, in which the users that are allowed to have access to the application are determined dynamically.

  For more information about the dynamic runtime security mechanism, see "Dynamic Runtime Security" on page 28.

For information about how to enable SSO in your Application Platform projects by adding SSO to the OSGi service layer, see *webMethods Application Platform User's Guide*.

Class and Description
com.softwareag.applatform.security.SecurityContext A class that provides a set of methods that are backed by the internal authorization service. Before the target method is invoked, an instance of this class is injected in any field of the same type that is defined in the @Service and @Secure annotated class.

You can query the role and subject information for the currently logged user by using the following methods in the SecurityContext class:

Method Name	Return Value	Method Parameters	Description
isUserInRole	Boolean	String. The role name.	Checks if the current user has the given role.
isUserInRoles	Boolean	String or string list. An array of role names.	Checks if the current user has all the given roles.
currentSubject	`javax.security.auth.Subject`		Returns the JAAS subject representation of the current user.
getBackingSubject	`org.apache.shiro.subject.Subject`		Obtains the backing security

Method Name	Return Value	Method Parameters	Description
			instance of the user.
isAuthenticated	Boolean		Checks if the current user is authenticated.

Annotation and Description
com.softwareag.applatform.security.Secure A marker annotation that indicates that the Application Platform service is secured and requires an authenticated subject when its methods are invoked. This annotation is used together with the @Service annotation at the type or class level. For information about the @Service annotation, see "Publishing POJOs as OSGi Services" on page 10.

Declarative Security

Application Platform enables you to add declarative security to POJOs that are published as OSGi services by using the @Service annotation. To add security to POJOs that are published as OSGi services, you can use the @Secure annotation, together with a set of common Java EE security annotations. Application Platform supports the following common Java EE security annotations, which you can use at the class or method level:

- @DenyAll
- @PermitAll
- @RolesAllowed

The following sample code shows an implementation of declarative security in an OSGi service. The @Secure annotation indicates that the AdderService service is secure. By default, invocation of the service methods is denied with the @DenyAll annotation. The @RolesAllowed annotation allows invocation of the add method by users with `Admin` or `Developer` role.

```
@Service
@Secure
@DenyAll
public class AdderService {

     @RolesAllowed({"Admin", "Developer"})
     public float add(float x, float y) {
          return x + y;
     }
}
```

Dynamic Runtime Security

Application Platform enables you to implement dynamic runtime authentication and authorization, in which the roles allowed for a user are not known in advance. To add dynamic runtime security, you can use the SecurityContext class. If the SecurityContext field type is specified in the class and gets injected at runtime, you must add the @Secure annotation to the corresponding Application Platform POJO service class.

The following sample code shows an implementation of dynamic runtime security in an OSGi service, where:

- A POJO class, named GreeterImpl, implements the IGreeter interface. The IGreeter interface is marked as secure with the @Secure annotation.
- The @DenyAll annotation at the class level denies access to all methods at runtime.
- The @RolesAllowed annotation overrides the @DenyAll annotation for the greetMe method for users that have `Administrators` or `Developers` role.
- The logCurrentSubject method uses the secContext field to retrieve the Java Authentication and Authorization Service (JAAS) Subject representation of the currently logged in user. The secContext field is of type SecurityContext and it is injected at runtime before the logCurrentSubject method is invoked with a valid instance.
- After the logCurrentSubject method retrieves the JAAS Subject, it prints the instance details of the associated Principal.

```
@Service
@Secure
@DenyAll
public class GreeterImpl implements IGreeter {

    public static final String KEY_HELLO = "hello";
    private String key = KEY_HELLO;

    @ServiceReference(id = "resourceUtilRef", interfaces =
    { "com.example.osgi.greet.impl.ResourceUtil" })
    ResourceUtil resUtil;

//injected at method invocation time
    private SecurityContext secContext;

    @Override
    @RolesAllowed({ "Administrators", "Developers" })
    public String greetMe(String name) {
       logCurrentSubject();
       return greetMe(name, Locale.getDefault());
    }
private void logCurrentSubject() {
    Subject subj = secContext.currentSubject();
    if (subj != null) {
       Set<SagUserPrincipal> users = subj.getPrincipals(SagUserPrincipal.class);
       if (users != null) {
            for (SagUserPrincipal sup : users) {
            System.out.println("Current logged in user is " + sup.getName());
            }
       }
```

```
        } else {
             System.err.println("No authenticated subject found!");
        }
    }
}
```