Getting Started with the webMethods Application Platform API

Version 9.9

October 2015

WEBMETHODS

Document ID: AP-API-GS-99-20151015

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.

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 {
}
```

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.

Property	Default Value	Description
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[]** Optional. The list of interfaces, under which the service will be published. If you do not specify a value for this property, the service 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.

Properties of @Property

Property	Default Value	Description
key	`""`	**String** Required. The name or key of the property.
values	`{}`	**Sting[]** Required. The values to be associated with the property name.

Property	Default Value	Description
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 annotations 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.

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.

Property	Default Value	Description
interfaces	`{}`	**String[]** 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 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

Property	Default Value	Description
		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	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 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

Method Name	Return Type	Method Arguments	Description
			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 8.

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

The following table outlines the related API documentation:

Class	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 {
}
```

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.