Application Platform

API Guide

Version 9.8

April 2015

This document applies to webMethods Application Platform Version 9.8 and to all subsequent releases. Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

DOCUMENT ID: PLD-API-98-20150415

CONTENTS

1 About this Guide

Software AG Application Platform IDE components are installed as a set of features within Software AG Designer. Online help is included in Designer Guide node of the Eclipse Help table of contents. Expand this node to view the available help sets. If a feature is not installed, there will be no help set available for it. However, you can view PDF versions of all designer features on the Software AG Documentation website.

1.1 Document Conventions

Convention	Description
Bold	Identifies elements on a screen
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 text.
{ }	Indicates a set of choices from which you must choose one. Type only the information found inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of the choices only. 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 (…).

1.2 Online Information

This document and other Software AG documents mentioned in this guide may be found at the Software AG Documentation website.

2 Introduction

This document provides the API documentation for 9.8.0 (April 2015) release of webMethods Application Platform.

2.1 Publishing Plain Old Java Objects (POJOs) as OSGi services

2.1.1 @Service

This annotation is used to mark a POJO class to be exposed as an OSGi service. It should be specified on a class type. An example usage is as follows:

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

interface MyInterface {
}
```

Properties	Default Value	Type	Required	Description
name	Simple name of the annotated class	String	No	The name of the bean backing this service; if unspecified will default to the simple name of the bean class
value	Simple name of the annotated class	String	No	An alternate way to specify the name of the service bean; useful when not specifying any other attributes
ranking	0	Integer	No	The ranking value to be published as the 'service.ranking' property for this service to distinguish between
init	""	String	No	The method to invoke when the bean backing the service is initialized
destroy	""	String	No	The method to invoke when the bean backing the service is destroyed
interfaces	FQN of annotated class	String[]	No	The list of interfaces under which the service will be published; If not specified, the service will only be published under the name of the implementation class
dependsOn	""	String	No	Used to express a dependency on another component that must be fully initialized before

				this service can be initialized and exported
properties	{}	Property[]	No	The list of service properties to be published with the service

2.1.1.1 @Property

Declare the property for this service; There can be more than one value for the key and it can optionally specify the type of the key and the type of the values

Properties	Default Value	Type	Required	Description
key	""	String	Yes	The name or key of the property
values	{}	String[]	Yes	The values to be associated with the property name
valueType	java.lang.String	String	No	The type of the values of this property

Example

The example below shows the GreeterImpl POJO class registered as an OSGi service under the name "greeter-impl" and 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;
    }
}
```

2.2 Inject service dependencies into other service

2.2.1 @ServiceReference

This annotation is used to inject a service from the runtime registry into another service being published (using the @Service annotation). This provides a form of dependency injection

where there the injected dependency is another POJO/bean already published in the runtime as an OSGi service.

A setter method to set the injected POJO reference must be specified in the same class accompanying the field declaration (containing the @ServiceReference annotation)

The following are the configurable annotation properties:

Properties	Default Value	Type	Required	Description
id	""	String	Yes	An unique identifier for this service reference. The specified id must not conflict with any other implicit or explicit @Service annotation name attribute value
interfaces	{}	String[]	Yes (if filter not specified)	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	Yes (if interfaces not specified)	An OSGi filter expression that constrains the service registry lookup to only those services that match the given filter. The filter string is of the form "(property-name = value)". E.g. (asynchronous-delivery=true) restricts the service lookup to those services that have the property named asynchronous-delivery set to value true.
timeout	5000 ms	Integer	No	The amount of time (in milliseconds) to wait for a backing service to be available when an operation is invoked. If no matching service becomes available within the timeout period, an unchecked ServiceUnavailableException will be thrown

componentName	""	String	No	A convenient shortcut for specifying a filter expression that matches on the property named org.eclipse.gemini.blueprint.bean.name that is automatically advertised for beans published using the @Service annotation
dependsOn	""	String	No	Used to specify 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	No	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 also already available Note: It is an error to 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.

Example

The example below 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 {
        ...
}
```

2.3 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 to look up registered services.

The following are the public API methods in ServiceUtil class:

Name	Return type	Arguments	Description
getService	T	ServletContext servletCtx Class<T> serviceCls	Returns the instance of the OSGi service of type serviceCls from the given ServletContext. This method will look 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> serviceCls BundleContext bundleCtx	Get the OSGi service of given serviceCls type using the given BundleContext or null if there is no service of that type registered
getBundleContext	BundleContext	Class<?> bundleCls	Get the BundleContext from the bundle containing the given class or null
getService	T	Class<T> serviceCls	Get the OSGi service for given service class type, or null if there is no service of that type registered

2.3.1 Dynamic POJO service configuration

Application Platform provides the ability to dynamically configure a published POJO service (using the @Service annotation mentioned in section: Publishing POJOs as OSGi services)

Please see the section "Project Dynamic Configuration" in the Application Platform User's Guide for the steps to take to enable dynamic service configuration in Application Platform projects.

The table below outlines the related API documentation:

Class	Description
org.osgi.service.cm.ManagedService	See OSGi v4.3 javadoc: http://www.osgi.org/javadoc/r4v43/cmpn/org/osgi/service/cm/ManagedService.html

The following methods must be implemented from the ManagedService interface:

Name	Return type	Arguments	Description
update	void	java.util.Dictionary<java.lang.String,?> properties	See OSGi v4.3 javadoc for updated method: http://www.osgi.org/javadoc/r4v43/cmpn/org/osgi/service/cm/ManagedService.html#updated(java.util.Dictionary)

2.4 Exposing POJO classes as IS assets

2.4.1 @ExposeToIS

This annotation is used to identify a class containing one or more methods to be exposed as IS services. It is combined with the @Service and @ExposedMethod annotations to support presentation of methods in a Java POJO as IS services. Because the generated IS assets assume that the Java class is registered in OSGi as a service, this annotation must be used with the @Service annotation. This Java fragment demonstrates use of the @ExposeToIS annotation:

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

This annotation has one optional property.

Properties	Default Value	Type	Required	Description
packageName	""	String	No	The name of the IS package where services from this class will be created. Note that this is the name of an IS package, not a Java package. If no value is supplied, at IS generation time, the value of the @Service.name property will be used for IS package name.

2.4.2 @ExposedMethod

This annotation identifies a method to be exposed as an IS service. It is valid only on public methods. Since IS 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.

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

This annotation has no properties.

Example

In this example, the OrdersServiceImpl class implements the OrdersService interface which declares several methods, including the two exposed here. The result of publishing this POJO in an Application Platform project will be creation of several artifacts in the IS namespace.

As a result of the 'packageName' property, an IS package named 'OrdersService' will be created if necessary. From the name of the Java package where the OrdersService interface is defined, the new IS package will contain a folder named 'com.softwareag.demp.orders.api'.

Each of the exposed methods will create an IS service in the new folder. The service name will match the exposed method name. The signatures for these new IS services will match the method signatures For example, the orderReceipt service signature will include a String output and one input, named inItem, of type Document, where the document structure matches the properties from 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 were omitted from this example code, then the package name in the IS namespace would be 'RegisteredOrdersService', from the @Servce annotation.