Getting Started with the webMethods Application Platform API

Version 10.3

October 2018

Document ID: AP-API-GS-103-20200110

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 service names and locations in the format *folder.subfolder.service* , APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	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 and Support

Software AG Documentation Website

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

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to “empower@softwareag.com” with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at “https://empower.softwareag.com/”.

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”.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at “https://empower.softwareag.com/public_directory.asp” and give us a call.

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.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

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.
- Invoke IS services in web applications.

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 describes the properties of @Service and specifies the default value for each property.

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	**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
	(FQN) of the annotated class	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 describes the properties of @Property and specifies the default value for each 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 describes the properties of @ServiceReference and specifies the default value for each property.

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`.

Property	Default Value	Description
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 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

The following table describes the class you can use to look 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 describes the public API methods in the ServiceUtil class and specifies the return type and method arguments for each method type.

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.

Method Name	Return Type	Method Arguments	Description
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 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 12.

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

The following table describes the class you can use to dynamically configure a published POJO service.

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

The following table describes the public API methods in the ManagedService class and specifies the return type and method arguments for each method type.

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 describes the properties of @ExposeToIS and specifies the default value for each property.

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.

Property	Default Value	Description
		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".

The following table lists and describes the available classes in the Application Platform integration test framework.

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

Class and Description
A custom JUnit runner that supports running parameterized tests in the Application Platform integration test framework.

The following table lists and describes the available annotations 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.
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 implementation, which can be overridden, for the methods, listed in the following table. The table shows the annotation name for each method name.

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 JMX 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 implementation, which can be overridden, for the methods, listed in the following table. The table shows the annotation name for each method name.

Annotation Name	Method Name
@BeforeClass	setupClass
@AfterClass	destroyClass

Annotation Name	Method Name
@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 describes the properties of @TestBundle and specifies the default value for each property.

Property Name	Default Value	Description
symbolicName	The project name of the corresponding Application	**String** Required. The symbolic name of the bundle that contains this test class when it is deployed to the configured server runtime.

Property Name	Default Value	Description
	Platform project.	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 describes the properties of @RunOnServer and specifies the default value for each property.

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

Application Platform provides the following methods for securing the web application layer:

- Using a set of valves to implement security constraints.
- Using a security filter, which you can add in the web.xml file.

You can use one or both of these methods to secure your web applications. When a request is sent to the Tomcat run-time, the security filter is invoked last, while the valve plug-ins are executed in the middle of the request processing.

Valve-Based Security

Valves are plug-ins that you can use to access inbound request messages before they reach the security filter. You can use a set of valves in order to implement a `<security-constraint>` element. For information about security constraints, see the Oracle documentation. For information about valves, see the Apache Tomcat documentation.

To implement valve-based security using Application Platform, you must configure the security realms you require and declare security restrictions in your WAR projects.

Configuring Security Realms

Application Platform maps WAR security to Integration Server by using Tomcat realms. When you create Integration Server instances, you must add the required security realms to the server.xml file, located in *Software AG_directory*profiles/IntegrationServer/configuration/tomcat/server.xml and map them to the AppPlatformRealm. By default, only the LockOutRealm is added to the server.xml file.

To map the LockOutRealm to the AppPlatformRealm, add the following code to server.xml:

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
   <Realm className="com.softwareag.applatform.pls.security.jaas.AppPlatformRealm"
      name="AppPlatformRealm"
      userClassNames="com.softwareag.security.jaas.principals.SagUserPrincipal"
      roleClassNames="com.softwareag.security.jaas.principals.SagRolePrincipal"
      defaultRealm="AppPlatformRealm"/>
   </Realm>
```

Application Platform enables you to implement client-certificate SSL authorization in your WAR projects. To enable client-certificate SSL authorization, you must update the META-INF/context.xml file your WAR project directory by adding the following code:

```
<!--Alternatively, place this code in the server, host, or engine configuration.-->
<!--If the name is not correct, the web page will fail with an invalid-->
<!--configuration error 60 seconds after the service tracker gives up-->
<!--com.softwareag.platform.catalina.auth.SINRealm-->
<Realm className="com.softwareag.applatform.pls.security.jaas.AppPlatformRealm"
   name="AppPlatformRealm"
   appName="AppPlatformRealm"
   userClassNames="com.softwareag.security.jaas.principals.SagUserPrincipal"
   roleClassNames="com.softwareag.security.jaas.principals.SagRolePrincipal"
   defaultRealm="AppPlatformRealm"/>
```

Declaring Security Restrictions in WAR Projects

To declare the required security restrictions in your WAR project, do one of the following:

- Use the annotations provided by Application Platform. For more information, see "OSGi Service Layer Security" on page 30.
- Add a web.xml configuration file containing security constraints to your project. For more information about implementing security constraints using the web.xml file, see the Oracle documentation.

Filter-Based Security

Application Platform provides a security filter that you can 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*.

The following table describes the security filter that you can add in the web.xml file.

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 describes the properties of AppPlatformSecurityFilter and specifies the default value for each property.

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. ■ `FORM` Form authentication. Requires attributes for user name and the password, as follows: j_username j_password	**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.
roleNamesAllowed	Depends on the runtime server, as follows:	**String List** Optional. A comma-separated list of allowed user roles. Users are authenticated

Property	Value	Description
	■ 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*.	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.

Session Concurrency Filter

The *session concurrency filter* is a java servlet filter that serializes the requests from a WAR project to a server over one session. You can use the session concurrency filter to:

- Reduce the load imparted on the server by highly concurrent web applications.
- Resolve errors that might occur because of logic that is not thread safe.
- Potentially improve performance in cases when serializing requests is faster than running the requests concurrently.

The session concurrency filter is intended for web applications that utilize HTML <FRAMESET> or <IFRAME> tags.

To use the servlet filter, you must update the web.xml file of your application with the following filter:

```
<filter>
       <filter-name>maxThreadsPerSession</filter-name>
       <display-name>
                com.softwareag.applatform.pls.is.web.filter.SessionConcurrencyFilter
       </display-name>
```

```
        <description>Serializes requests for each Session</description>
        <filter-class>
                 com.softwareag.applatform.pls.is.web.filter.SessionConcurrencyFilter
        </filter-class>
        <!-- optional element. The default is '1' -->
        <init-param>
                        <param-name>maxPerSession</param-name>
                        <param-value>1</param-value>
        </init-param>
</filter>
<filter-mapping>
    <filter-name>maxThreadsPerSession</filter-name>
    <url-pattern>*/*</url-pattern>
</filter-mapping>
```

OSGi Service Layer Security

The Application Platform API provides a class 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 32.

- 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 33.

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*.

The following table describes the class and annotations that are provided by Application Platform for implementing security at the OSGi service layer.

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 available methods in the SecurityContext class. The following table describes the public API methods in the SecurityContext class and specifies the return type and method parameters for each method type.

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 instance of the user.
isAuthenticated	Boolean		Checks if the current user is authenticated.

The following table lists and describes the annotations you can use to implement OSGi service layer security using the Application Platform API.

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 12.
com.softwareag.applatform.security.AclAllowed A marker annotation that takes a single Access Control List (ACL) value as a parameter. Use this annotation to define ACLs for Integration Server.Use this annotation at the class or method level, as follows: ■ When added at class level, it applies to all methods.

Annotation and Description
■ When added at method level, it applies only to that method and overrides any class-level value. ■ The value of the @AclAllowed annotation corresponds to the Integration Server ACL to which the current user belongs. The ACL value must be already present in Integration Server when the log-in request is sent to the secured service.

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 for My webMethods Server
- @AclAllowed for Integration Server

The following sample codes show an OSGi service implementation of declarative security, where the @Secure annotation indicates that the AdderService service is secure and the invocation of the service methods is denied by default with the @DenyAll annotation. The examples show how you can allow invocation of the add method, as follows:

- By users with `Admin` or `Developer` role in My webMethods Server, by using the @RolesAllowed annotation:

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

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

- By users with `Developer` role in Integration Server, by using the @AclAllowed annotation:

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

     @AclAllowed({"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!");
        }
    }
}
```

Invoking IS Services in Web Applications

Application Platform enables you to invoke IS services:

- From JSP pages.
- From servlets.
- Anonymously.

Invoking IS Services from JSP Pages

You can configure your Tomcat instance to control when JSP pages are compiled into Java classes. When you develop JSP pages, you can configure them to invoke IS services using:

- A webMethods tag library for JSP.

  For information about the webMethods JSP tag library, see " webMethods Tag Library Summary" on page 40.

- Direct service invocation using the IS service API.

  For information about the IS service API, see the Integration Server Java API Reference.

Direct Service Invocation

You can invoke IS services by using the `com.wm.app.b2b.server.Service.doInvoke()`, provided by Integration Server, in a JSP page scriptlet block.

Important: If you have migrated a web application from the WmTomcat package to the WmAppPlat package, you must implement an additional logic to all JSP pages to ensure that the executing thread contains a valid state before it invokes any IS services. This is required, since the Tomcat connectors now receive requests through the WmAppPlat package, instead of an HTTP dispatch handler. Software AG recommends that you use the `<webm:invoke>` custom tag and implement the following logic:

```
import com.wm.app.b2b.server.*;
import javax.servlet.*;
try{
// --- this additional code must be added to ensure the invoke state exists
```

```
   for the executing thread.
   InvokeState invokeState = InvokeState.getCurrentState();
      if ( invokeState == null) {
            InvokeState.setCurrentState(InvokeState.create( ));
      }
// ---
   IData idata = Service.doInvoke( "pub.string", "concat", idata );
 }catch( Throwable t){
    throw new ServletException(t);
 }
```

Invoking IS Services from Servlets

You can use the Integration Server API to invoke IS service instances from the source code of your web applications. For example:

```
import com.wm.app.b2b.server.*;
import javax.servlet.*;
try{
   InvokeState invokeState = InvokeState.getCurrentState();
      if ( invokeState == null) {
            InvokeState.setCurrentState(InvokeState.create( ));
      }
   IData idata = Service.doInvoke( "pub.string", "concat", idata );
 }catch( Throwable t){
    throw new ServletException(t);
 }
```

Note: This way of invoking IS services is not recommended, as it uses the Integration Server API directly and it might require additional updates to your applications when you upgrade to higher Software AG product versions.

Anonymous Invocation

By default, unauthenticated JSP page invocations use a `Default` username. To update this username, open the *SoftwareAG_directory* /profiles /*IS_ instance* /configuration/com.softwareag.platform.config.propsloader/ com.softwareag.applatform.pls.service.executor.pid.properties file and update `defaultUserId=Default`, as required.

3 webMethods Tag Library for JSP

Overview

The webMethods tag library for JSP is a set of customized implementation classes. You use tags in this library in JSPs to import actions that let you call Integration Server services and retrieve data from the Integration Server pipeline.

The examples used in this section are based on a specific pipeline. The following table displays the key-value pairs from the pipeline.

Key	**Value**	
submitter	Mark Asante	
shipNum	991015-00104	
shipDate	10/15/99	
carrier	UPS	
serviceLevel	Ground	
arrivalDate	10/18/99	
items	**Key**	**Value**
	qty	10
	stockNum	BK-XS160
	description	Extreme Spline 160 Snowboard- Black
	orderNum	GSG-99401088
	status	Partial Order
	qty	15
	stockNum	WT-XS160
	description	Extreme Spline 160 Snowboard- White
	orderNum	GSG-99401088

Key	**Value**	
	status	Complete
supplierInfo	**Key**	**Value**
	companyName	Bitterroot Boards, LLC
	streetAddr1	1290 Antelope Drive
	streeAddr2	
	city	Missoula
	state	MT
	postalCode	59801
	supplierID	BRB-950817-001
	phoneNum	406-721-5000
	faxNum	406-721-5001
	email	Shipping@BitterrootBoards.com
buyerInfo	**Key**	**Value**
	companyName	Global Sporting Goods, Inc.
	accountNum	
	phoneNum	(216) 741-7566
	faxNum	(216) 741-7533
	streetAddr1	10211 Brookpark Road
	streetAddr2	
	city	Cleveland
	state	OH

Key	Value	
	postalCode	22130
	email	Receiving@GSG.com
backitems	SL-XS170 Extreme Spline 170 Snowboard- Silver	
	BL-KZ111 Kazoo 111 Junior Board- Blue	
	BL-KZ121 Kazoo 121 Junior Board- Blue	

webMethods Tag Library Summary

This section provides a summary of the tags in the webMethods tag library.

Note: Tags are case sensitive. In your JSP, you must type them exactly as shown in this section (for example, `<webm:comment>`, not `<WEBM:COMMENT>` or `<Webm:Comment>`).

The following table lists and describes each tag from the webMethods tag library.

Tag	Description
`<jsp:include>`	Inserts a text file in the JSP. At run time, Integration Server inserts the contents of the specified file in the JSP and processes the tags that the file contains. For more information, see "<jsp:include>" on page 42.
`<webm:comment>`	Adds a comment to the JSP. For more information, see "<webm:comment>" on page 43.

`<webm:ifvar>`
`<webm:then>`
`<webm:else>`
	Executes a block of code in the JSP if a specified condition is met. The condition can be either that a certain variable exists or that a variable has a particular value in the Integration Server pipeline. You can also define a block of code to be executed if the specified condition is not met. For more information, see "<webm:ifvar>" on page 43.

`<webm:invoke>`
`<webm:onError>`
	Calls an Integration Server service. You can also define different blocks of code to be executed if the service completes without errors and if the

Tag	Description
`<webm:onSuccess>`	service fails. Integration Server runs the specified service at run-time and returns the results of the service to the JSP. For more information, see "<webm:invoke>" on page 45.
`<webm:loop>` `<webm:loopsep>`	Executes a block of code in the JSP once for each element in a specified Document, Document List, or String List in the Integration Server pipeline. You can insert a specific character sequence between the results from the <webm:loop> tag in the HTML page produced from the JSP. For more information, see "<webm:loop>" on page 46.
`<webm:nl>`	Generates a newline character on the HTML page produced from the JSP. For more information, see "<webm:nl>" on page 50.
`<webm:rename>`	Renames or copies a variable in the Integration Server pipeline. For more information, see "<webm:rename>" on page 51.
`<webm:scope>`	Limits the variables in the Integration Server pipeline that are available for a specified block of code in the JSP. For more information, see "<webm:scope>" on page 51.
`<webm:switch>` `<webm:case>`	Executes one of multiple blocks of code, based on the value of a variable in the Integration Server pipeline. For more information, see "<webm:switch>" on page 54.
`<webm:sysvar>`	Inserts a special variable or server property into the HTML page produced from the JSP. For more information, see " <webm:sysvar>" on page 56.
`<webm:usePipeline>`	Transforms the current Integration Server pipeline available in the Java code to an IData variable called webm_pipe. For more information, see "<webm:usePipeline>" on page 56.
`<webm:value>`	Inserts the values of one or more variables from the Integration Server pipeline into the

Tag	Description
	HTML page produced from the JSP. For more information, see "<webm:value>" on page 57.

<jsp:include>

Use the `<jsp:include>` tag to insert a text file in the JSP. At runtime, the Tomcat servlet container inserts the contents of the specified file in the JSP and processes the tags that the file contains.

Tip: If you use JSPs extensively, you might want to build a library of standard "code fragments" that you reference using `<jsp:include>` tags.

Syntax

For interpreted pages:	`<jsp:include page="file" flush="true"/>`
For static pages:	`<%@ include file="file"%>`

Arguments

Argument	Description
file.extension	Text file to insert. If the text file is not in the same directory as the JSP that references it, specify the path to the file relative to the JSP.

Examples

The following examples insert the TMPL_ShipToBlock.html file into the JSP.

- To insert the TMPL_ShipToBlock.html file in the same directory as the JSP.

```
<webm:scope recordName="buyerInfo">
<p>Shipped To:<br>
<jsp:include page="TMPL_ShipToBlock.html" flush="true"/>
</p>
</webm:scope>
```

- To insert the TMPL_ShipToBlock.html file in a subdirectory named "StandardJSPs" within the directory in which the JSP resides.

```
<webm:scope recordName="buyerInfo">
<p>Shipped To:<br>
<jsp:include page="StandardJSPs/TMPL_ShipToBlock.html" flush="true"/>
</p>
</webm:scope>
```

- To insert the TMPL_ShipToBlock.html file in the parent directory of the JSP.

```
<webm:scope recordName="buyerInfo">
<p>Shipped To:<br>
<jsp:include page="../TMPL_ShipToBlock.html" flush="true"/>
</p>
</webm:scope>
```

<webm:comment>

Use the `<webm:comment>` tag to add a comment to the JSP.

Syntax

```
<webm:comment>comment</webm:comment>
```

Example

```
<webm:comment>
Use this JSP to generate an order list from any document that contains
a purchase item number, quantity, description, and PO number.
</webm:comment>
```

Note: You can also use the HTML comment syntax `<!-- comment -->` to include comments in JSP code.

<webm:ifvar>

Use the `<webm:ifvar>` tag to execute a block of code in the JSP if a specified condition is met. The condition can be either that a certain Integration Server pipeline variable exists, or that the variable has a particular value. You can also define a different block of code to be executed if the specified condition is not met.

Use `<webm:then>` to define the block of code to be executed if the condition is met. Use `<webm:else>` to define the block of code to be executed if the condition is not met.

Syntax

```
<webm:ifvar variable="variable" [isNull="true"] [notEmpty="true"]
      [equals="any_string"] [vequals="ref_variable"] [matches="regular_exp"]>
    <webm:then>block_of_code</webm:then>
    [<webm:else>block_of_code</webm:else>]</webm:ifvar>
```

Arguments

Argument	Description
variable	Pipeline variable to evaluate. **Example** `<webm:ifvar variable="carrier">`

Argument	Description
`[isNull="true"]`	Specifies that the condition is true if the variable exists but its value is null. **Example** `<webm:ifvar variable="carrier" isNull="true">`
`[notEmpty="true"]`	Specifies that the condition is true if the variable contains one or more characters and false if the value of the variable is null. Use for string variables only. **Example** `<webm:ifvar variable="carrier" notEmpty="true">`
`[equals="`*`any_string`*`"]`	Specifies that the condition is true if the variable matches the specified string. *any_string* is case sensitive. For example, FedEx does not match Fedex or FEDEX. **Example** `<webm:ifvar variable="carrier" equals="FedEx">`
`[vequals="`*`ref_variable`*`"]`	Specifies that the condition is true if the variable matches the value of the specified pipeline variable. **Example** `<webm:ifvar variable="supplierInfo/state" vequals="buyerInfo/state">`
`[matches="`*`regular_exp`*`"]`	Specifies that the condition is true if the variable matches the specified regular expression . **Example** In the following example the condition is true if the value of the pipeline variable carrier starts with UPS: `<webm:ifvar variable="carrier" matches="UPS*">`

Example

If a variable named *AuthCode* exists in the pipeline, the following code inserts a success message into the HTML page produced from the JSP. The success message includes the current date and the value of the pipeline variable *AuthCode* . If the variable does not exist, the code inserts an error message.

```
<webm:ifvar variable="AuthCode">
    <webm:then>
         <p>Authorization code received <webm:sysvar variable="date">
         <webm:value variable="AuthCode"></p>
    </webm:then>
    <webm:else>
        <p>Error: Authorization code does not exist.</p>
    </webm:else>
</webm:ifvar>
```

<webm:invoke>

Use the `<webm:invoke>` tag to call an Integration Server service. You can also define a block of code to be executed if the service completes without errors and a block of code to be executed if the service fails. Integration Server runs the specified service at run time and returns the results of the service to the JSP.

Use the `<webm:onSuccess>` tag to define the block of code to be executed if the service completes without errors. Use the `<webm:onError>` tag to define the block of code to be executed if the service fails. You can use pipeline variables in the `<webm:onError>` tag. The following table lists the available pipeline variables:

Key	Description
errorClass	Exception name.
errorMessage	Error message text.
localizedMessage	Localized error message text that uses the locale of the customer.
errorService	Failed service.
errorInputs	Input for the failed service.
errorOutputs	Output for the failed service. Can be null.

Syntax

```
<webm:invoke serviceName="service">
    <webm:onSuccess>block_of_code</webm:onSuccess>
    <webm:onError>block_of_code</webm:onError>
</webm:invoke>
```

Arguments

Argument	Description
service	Fully qualified name of the service to call.

Example

To call the Integration Server service orders:getShipInfo, which gets shipping information from a back-end system and builds an HTML form that allows the user to edit or cancel an order and display error information if the service fails:

```
<webm:invoke serviceName="orders:getShipInfo">
<h2>Shipping Details for Order  <webm:value variable="/oNum"/></h2>
<p>Date Shipped: <webm:value variable="shipDate"/><br>
Carrier: <webm:value variable="carrier"/>
<webm:value variable="serviceLevel"/></p>
<hr>
<webm:ifvar variable="shipDate" notempty="true">
<form action="http://rubicon:5555/orders/editShipInfo.dsp"
    method="get">
<p><b>Change this Shipment:</b></p>
<p><input type="RADIO" name="action" value="edit">
    Edit Shipment Details</p>
<p><input type="RADIO" name="action" value="cancel">
    Cancel this shipment</p>
    <input type="SUBMIT" value="Submit">
    <input type="HIDDEN" name="oNum"
    value="<webm:value variable='/oNum'/>"
</form>
    <hr>
</webm:ifvar>
<p><a href="http://rubicon:5555/orders/getorder.dsp
?action=showorder&oNum=<webm:value variable='/oNum'/>View Entire Order</a></p>
<webm:onError>
<hr>
<p><font color="#FF0000">Integration Server could not process
    your request because the following error occurred. Contact
    your server administrator.</font></p>
<table width="50%" border="1">
    <tr>
        <td><b>Service</b></td>
        <td><webm:value variable="errorService"/></td>
    </tr>
    <tr>
        <td><b>Error</b></td>
        <td><webm:value variable="errorMessage"/></td>
     </tr>
</table>
<hr>
</webm:onError>
</webm:invoke>
```

<webm:loop>

Use the `<webm:loop>` tag to execute a block of code once for each element in a specified Document, Document List, or String List in the Integration Server pipeline.

If you do not specify a Document, Document List, or String List to loop over, Integration Server will loop over the entire pipeline.

You can use the `<webm:loopsep>` tag to insert a specified character sequence between the results from the `<webm:loop>` tag in the HTML page produced from the JSP.

`<webm:loopsep>` does not insert the character sequence after the result produced by the last loop iteration.

At run time, Integration Server loops over the pipeline data. The following table describes the Integration Server loop for each data structure.

For this data structure	Integration Server loops over
Pipeline	Each key in the pipeline.
Document	Each key in the Document.
Document List	Each Document in the Document List.
String List	Each String in the String List.

Syntax

For the pipeline:

```
<webm:loop loopStruct="true" [excludePrivate="true"]>
block_of_code
    [<webm:loopsep sepString="separator_string"/>]
</webm:loop>
```

For a Document:

```
<webm:loop variable="loop_variable" loopStruct="true" [excludePrivate="true"]>
block_of_code
    [<webm:loopsep sepString="separator_string"/>]
</webm:loop>
```

For a Document List or String List:

```
<webm:loop variable="loop_variable" [loopStruct="true"]>
block_of_code
    [<webm:loopsep sepString="separator_string"/>]
</webm:loop>
```

Arguments

Argument	Description
`[variable="loop_variable"]`	Document, Document List, or String List to loop over. If you omit this option, Integration Server loops over the entire pipeline. **Example** `<webm:loop variable="items">`
`[loopStruct="true"]`	Sets Integration Server to loop over each key in the pipeline, Document, Document List, or String List.

Argument	Description
	Example `<webm:loop variable="items" loopStruct="true">`
`[excludePrivate="true"]`	Sets Integration Server to skip keys with names starting with `$`. **Example** `<webm:loop variable="items" excludePrivate="true">`

Usage Notes

- To access the names of the keys dynamically in the structure over which you are looping when you are using `[loopStruct="true"]`, use the predefined *$key* keyword.
- If you are using `[loopStruct="true"]` for nested loops and the outer loop is looping over a structure with dynamic contents and you require the inner loop to loop over the keys in the current key of the outer loop, set variable in the inner loop to `#$key` keyword. In most cases this keyword is used to process a set of Documents contained within another Document. See the Examples section.

Examples

- To add shipping information for each Document from Document List *items* in the HTML page produced from the JSP:

```
<p>This shipment contains the following items:</p>
<webm:loop variable="items">
    <p>
    <webm:value variable="qty"/>
    <webm:value variable="stockNum"/>
    <webm:value variable="description"/>
    <webm:value variable="status"/>
    </p>
  </webm:loop>
```

- To list each element from the String List *backItems* on separate lines in the HTML page and indicate that the elements are backordered:

```
<p>The following items are backordered</p>
<p>
<webm:loop variable="backitems">
    <webm:value/><BR>
</webm:loop>
</p>
```

- To list the elements in the String List *backItems* with commas between each elements and indicate that the elements are backordered:

```
<p>The following items are backordered</p>
<p>
<webm:loop variable="backitems">
    <webm:value/>
    <webm:loopsep sepString=","/>
```

```
</webm:loop>
</p>
```

- To display the names and values of the keys in the Integration Server pipeline on the HTML page:

```
<webm:loop loopStruct="true">
    <webm:value variable="$key"/><br>
    <webm:value/><br>
</webm:loop>
```

- To loop over the keys in the Document named *Addresses* which contents are not known before run-time and insert each name and its associated address into the HTML page produced from the JSP:

```
<p>Customer Addresses:</p>
<p>
<webm:loop variable="Addresses" loopStruct="true">
     <webm:value variable="$key"/>
     <webm:loop variable="#$key" loopStruct="true">
        <webm:value variable="streetAddr1"/>
        <webm:value variable="streetAddr2"/>
        <webm:value variable="city"/>
        <webm:value variable="state"/>
        <webm:value variable="postalCode"/>
    </webm:loop>
</webm:loop>
</p>
```

The following table shows the structure and key-value pairs used in the Document for the current example:

Key	Value	
Global Sporting Goods, Inc.	**Key**	**Value**
	streetAddr1	10211 Brookpark Road
	streetAddr2	
	city	Cleveland
	state	OH
	postalCode	22130
Fitness Shoes	**Key**	**Value**
	streetAddr1	2549 Oak Avenue
	streetAddr2	

Key	Value	
	city	Silver Spring
	state	MD
	postalCode	20905
Yoga Wear	Key	Value
	streetAddr1	6666 Maple Street
	streetAddr2	Suite 100
	city	New York
	state	NY
	postalCode	11302

<webm:nl>

Use the `<webm:nl>` tag to insert a newline character in the HTML page produced from the JSP.

Use this tag when you want to preserve the ending of a line that ends in a tag. This tag does not insert the HTML line break, `<br>`. It inserts a line break character, which Integration Server treats as white space. If you do not explicitly insert this tag on such lines, Integration Server drops the newline character following that tag.

Syntax

```
<webm:nl/>
```

Example

To insert the values of the variables *carrier* , *serviceLevel* , and *arrivalDate* on separate lines in the HTML page:

```
<hr>
<p>Shipping Info:
<webm:value variable="carrier"/><webm:nl/>
<webm:value variable="serviceLevel"/><webm:nl/>
<webm:value variable="arrivalDate"/><webm:nl/></p>
<hr>
```

<webm:rename>

Use the `<webm:rename>` tag to rename or copy a variable in the Integration Server pipeline.

Syntax

```
<webm:rename sourceVar="source_variable " targetVar="target_variable "
 [copy="true"]/>
```

Arguments

Argument	Description
source_variable	Pipeline variable to rename to or copy. *source_variable* can reside in any existing scope or Document.
target_variable	Pipeline variable to rename or copy *source_variable* . *target_variable* must be in the current scope. If *target_variable* does not exist, Integration Server creates it. If *target_variable* already exists, Integration Server overwrites it.
`[copy="true"]`	Tells Integration Server to copy *source_variable* to *target_variable* . By default, Integration Server renames *source_variable* to *target_variable* by copying *source_variable* to *target_variable* and then deleting *source_variable* .

Example

To rename the pipeline variable *state* in the Document *buyerinfo* to *ST* :

```
<webm:scope recordName="buyerInfo">
    <webm:rename sourceVar="state" targetVar="ST"/>
    <jsp:include page="TMPL_ShipToBlock.html" flush="true"/>
</webm:scope>
```

<webm:scope>

Use the `<webm:scope>` tag to limit the variables in the Integration Server pipeline that are available for a specified block of code in the JSP.

The specified scope remains in effect until Integration Server encounters the next `</webm:scope>` tag.

Syntax

```
<webm:scope [recordName="document "]
```

```
[options="param(name='value')
param(stringList[]='value1', 'value2',...'valuen')
rparam(document={name1='value1';name2='value2';...namen='valuen'})
rparam(documentList[]={name1='value1';name2='value2';...namen='valuen'}|
{name1='value1';name2='value2';...namen='valuen'})"]>  block_of_code
</webm:scope>
```

Arguments

Argument	Description
`[recordName="document"]`	Specifies the document to use as the current scope.
`[options="param(name='value')"]`	Adds to the current scope a String named *name* whose value is *value*. **Example** `<webm:scope options="param(csClass=Gold) param(csName='Joe Smith')"/>`
`[options="param(stringList[]='value1', 'value2',... 'valuen')"]`	Adds to the current scope a String List named *stringList* whose values are *value1 value2 ...valuen*. **Example** `<webm:scope options="param(shipPoints[]=BOS,LAX,NYC,PHL) param(warehouseLoc[]='Los Angeles',Philadelphia)"/>`
`[options="rparam(document={ name1='value1'; name2='value2';... namen='valuen'})"]`	Adds to the current scope a Document named document whose variables are name and whose values are *value*. **Example** `<webm:scope options="rparam(custServiceInfo= {csClass=Gold; csPhone='800-444-2300'; csRep='Ann Johnson'}) rparam(buyerInfo={buyerName='Joe Smith'; buyerPhone='800-333-1234'})"/>`
`[options="rparam(documentList[]={ name1='value1'; name2='value2';... namen='valuen'}| {name1='value1'; name2='value2';... namen='valuen'})"]`	Adds to the current scope a Document List named *documentList* whose variables are *name* and whose values are *value*. **Example** `<webm:scope options="rparam(custServiceCtrs[]= {csName=Memphis;csPhone='800-444-2300'}| {csName=Troy;csPhone='800-444-3300'}| {csName=Austin;csPhone='800-444-4300'}) rparam(custServiceReps[]={csRep='Ann Johnson';csExt=27}| {csRep='John Jones';csExt=28}| {csRep='Chris Smith';csExt=29})"/>`

Usage Notes

- If you specify multiple options, use a space to separate the options.
- If the value of a variable contains spaces, enclose the value within single quotes.
- If you set the value of a variable with one of the options, the value that you specify will replace the current value of that variable.
- When you use the `<webm:scope>` tag in a JSP, the entire tag must appear on one line.

Examples

- To code set a scope to the Document *buyerInfo* and insert the information from the scope into the HTML page produced by the JSP:

```
<webm:scope recordName="buyerInfo">
    <p>Shipped To:<br>
        <webm:value variable="companyName"/><br>
        <webm:value variable="streetAddr1"/><br>
        <webm:value variable="streetAddr2"/><br>
        <webm:value variable="city"/>
        <webm:value variable="state"/>
        <webm:value variable="postalCode"/>
    </p>
</webm:scope>
```

- To set a scope to the Document *buyerInfo*, add variables named *buyerClass* and *shipPoint* to the scope, and insert the information from the scope into the HTML page produced by the JSP:

```
<webm:scope recordName="buyerInfo"
 options="param(buyerClass=Gold) param(shipPoint='BWI Hub')">
    <p>Shipped To:<br>
        <webm:value variable="companyName"/><br>
        <webm:value variable="streetAddr1"/><br>
        <webm:value variable="streetAddr2"/><br>
        <webm:value variable="city"/>
        <webm:value variable="state"/>
        <webm:value variable="postalCode"/>
    </p>
    <hr>
    <p>Point of Departure: <webm:value variable="shipPoint"/><br>
    Customer Class: <webm:value variable="buyerClass"/></p>
</webm:scope>
```

- To set a scope to the Document *buyerInfo*, add variables named *buyerClass* and *shipPoint* from a Document named *shipInfo* to the scope, and insert the information in the scope into the HTML page produced by the JSP:

```
<webm:scope recordName="buyerInfo"
options="rparam(shipInfo={buyerClass=Gold;shipPoint='BWI Hub'})">
    <p>Shipped To:<br>
        <webm:value variable="companyName"/><br>
        <webm:value variable="streetAddr1"/><br>
        <webm:value variable="streetAddr2"/><br>
        <webm:value variable="city"/>
        <webm:value variable="state"/>
        <webm:value variable="postalCode"/>
    </p>
    <hr>
    <p>Point of Departure: <webm:value
    variable="shipInfo/shipPoint"/><br>
    Customer Class: <webm:value variable="shipInfo/buyerClass"/></p>
</webm:scope>
```

- To set a scope to the Document *buyerInfo*, adds a variable named *shipPoints* to the scope, add variables named *name* and *ssid* from a Document List called *custInfo*, and insert the information in the scope into the HTML page produced by the JSP:

```
<webm:scope recordName="buyerInfo"
options="param(shipPoints[]=DC,VA,LA,MD)
rparam(custInfo[]={name=Joe;ssid=ssid1}|{name=John;ssid=ssid2})">
<h3>Ship Points: </h3>
    <webm:loop variable="shipPoints">
    <webm:value /><br>
    </webm:loop>
<h3>Customers: </h3>
    <webm:loop variable="custInfo">
    <webm:value variable="name" />, <webm:value variable="ssid" />   <br>
    </webm:loop>
</webm:scope>
```

<webm:switch>

Use the `<webm:switch>` tag to execute one of multiple blocks of code, based on the value of a variable in the Integration Server pipeline.

Use `<webm:case>` to define each value and the associated block of code to execute. Integration Server evaluates `<webm:case>` tags in the order they appear in the JSP. When a case is true, Integration Server executes the associated block of code, then exits the `<webm:switch>` structure.

To define a default case to be executed if the specified variable does not exist or if none of the other cases is true, specify a `<webm:case>` tag with no *switch_value*. The default case must be the last `<webm:case>` tag in the `<webm:switch>` tag.

Syntax

```
<webm:switch variable="variable">
    <webm:case value="switch_value1">block_of_code</webm:case>
    [<webm:case value="switch_value2">block_of_code</webm:case>...
    <webm:case value="switch_valuen">block_of_code</webm:case>]
    [<webm:case>block_of_code</webm:case>]
</webm:switch>
```

Arguments

Argument	Description
variable	Pipeline variable which value to evaluate.
switch_value	A value of *variable* that triggers Integration Server to execute the associated block of code. *switch_value* must match the value of

Argument	Description
	variable exactly. *switch_value* is case sensitive. For example, FedEx does not match Fedex or FEDEX.

Examples

- To insert different paragraphs into the HTML page produced from the JSP based on the value in the pipeline variable *carrier* :

```
<webm:switch variable="carrier">
    <webm:case value="FedEx">
        <p>Shipped via Federal Express <webm:value="serviceLevel"/>
        <webm:value="trackNum"/></p>
    </webm:case>
    <webm:case value="UPS">
        <p>Shipped via UPS <webm:value="serviceLevel"/></p>
    </webm:case>
    <webm:case value="Freight">
        <p>Shipped via <webm:value="transCompany"/><br>
           FOB: <webm:value="buyerInfo/streetAddr1"/><br>
          <webm:value="buyerInfo/streetAddr2"/><br>
          <webm:value="city"/>, <webm:value="state"/>
          <webm:value="postalCode"/></p>
    </webm:case>
</webm:switch>
```

- To call different Integration Server services based on the value of the pipeline variable *action* :

```
<html>
    <head>
    <title>Order Tracking System</title>
    </head>
    <body bgcolor="#FFFFCC">
    <h1>Order Tracking System</h1>
    <hr>
    <webm:switch variable="action">
        <webm:case value="shipinfo"/>
            <webm:invoke serviceName="orders:getShipInfo"/>
            .
            .
            .
        <webm:case value="showorder"/>
            <webm:invoke serviceName="orders:orders:getOrderInfo"/>
            .
            .
            .
            .
        <webm:case value="showinvoice"/>
            <webm:invoke serviceName="orders:orders:getInvoices"/>
            .
            .
            .
    </webm:switch>
    <hr>
    <jsp:include page="stdFooter.txt" flush="true"/>
    </body>
</html>
```

<webm:sysvar>

You use the `<webm:sysvar>` tag to insert a special variable or server property into the HTML page produced from the JSP.

Syntax

```
<webm:sysvar variable="system_variable"/>
```

Arguments

Argument	Description
system_variable	System variable or property to insert. Valid values: ■ `host` - name of the Integration Server that processed the JSP ■ `date` - current date in the format “Weekday Month Day HH:MM:SS Locale Year”. For example, "Fri Aug 12 04:15:30 Pacific 1999". ■ `property(`*`property`*`)` - current value of the Integration Server property *property* . For example, `watt.server.port` or a Java system property like `java.home`. For a list of properties, see the *webMethods Integration Server Administrator’s Guide*.

Examples

- To insert the name of the Integration Server that processed the JSP into the HTML page:

  ```
  Response generated by host <webm:sysvar variable="host"/>
  ```

- To insert the value of the Integration Server property `watt.server.port`, which identifies the main HTTP listening port of Integration Server into the HTML page:

  ```
  <p>
  <webm:sysvar variable="host"/> was listening on
  <webm:sysvar variable="property(watt.server.port)"/>
  </p>
  ```

<webm:usePipeline>

Use the `<webm:usePipeline>` tag to make the current Integration Server pipeline available to the JSP in Java code as an IData variable named *webm_pipe* .

Syntax

```
<webm:usePipeline>block_of_code</webm:usePipeline>
```

Example

To print the contents of the current pipeline in a single long string to the HTML page produced from the JSP:

```
<webm:usePipeline>
<% System.out.println ("pipeline is:" + webm_pipe.toString()); %>
</webm:usePipeline>
```

<webm:value>

Use the `<webm:value>` tag to insert the values of one or more variables from the Integration Server pipeline into the HTML page produced from the JSP. You can also use the tag within a loop to insert the value of the current key from the loop by specifying the tag without any arguments.

Syntax

Separate nested fields with a forward slash (/).

```
<webm:value [variable="variable[/subvariable1/subvariable2/...
/subvariablen]"] [null="any_string"] [empty="any_string"] [index="n"]
[encode="code"] [decode="code"] [decimalShift="X"] [decimalShow="Y"]/>
```

Arguments

Argument	Description
`[variable="variable [/subvariable1/ subvariable2/.../ subvariable3]"]`	Pipeline variable which value to insert. Integration Server retrieves *variable* from the current scope. If you do not specify *variable* and you are inside a loop, Integration Server returns the value of the current key. You can select a variable out of the current scope by using the required syntax, as follows: ■ `/variable` - inserts the value of *variable* from the initial scope ■ `../variable` - inserts the value of *variable* from the parent of the current scope ■ `variable/ subvariable1/subvariable2/.../subvariable3`

Argument	Description
	- inserts the value of *subvariablen* from *variable* If you do not specify *variablen*, Integration Server assumes that the current element is within the current scope.
`[null="`*`any_string`*`"]`	When *variablen* is null, inserts *any_string*. **Example** `<webm:value variable="carrier" null="No Carrier Assigned"/>`
`[empty="`*`any_string`*`"]`	When *variablen* contains an empty string, inserts *any_string*. **Example** `<webm:value variable="description" empty="Description Not Found"/>`
`[index="`*`n`*`"]`	Inserts element *n* of the Document List or String List specified by *variable*. **Example** `<webm:value variable="backItems" index="1"/>`
`[encode="`*`code`*`"]`	Encodes the contents of *variable* before it is inserted. *code* specifies the encoding system to use, as follows: ■ `xml` - XML encoding ■ `b64` - Base-64 encoding ■ `url` - URL encoding
`[decode="`*`code`*`"]`	Decodes the contents of *variable* before it is inserted. *code* specifies the decoding system to use, as follows: ■ `b64` - Base-64 decoding ■ `url` - URL decoding

Argument	Description
`[decimalShift="`*`X`*`"]`	For decimal values only. Shifts the decimal point in the value of *variable X* positions to the right before inserting the value.
`[decimalShow="`*`Y`*`"]`	For decimal values only. Truncates the value of *variable* Y positions after the decimal before inserting the value.

Usage Notes

Use a forward slash (/) to separate nested fields.

Examples

- To call the orders:getOrderInfo service and insert the results of the service into the HTML page produced from the JSP:

```
<webm:invoke serviceName="orders:getOrderInfo"><br>
<p><webm:value variable="buyerInfo/companyName"/><br>
<webm:value variable="buyerInfo/acctNum"/>
    <p>This shipment contains the following items</p>
    <table width="90%" border="1">
        <tr>
            <td>Number</td><td>Qty</td>
            <td>Description</td><td>Status</td>
        </tr>
        <tr>
        <webm:loop variable="items">
            <td><webm:value variable="stockNum"/></td>
            <td><webm:value variable="qty"/></td>
            <td><webm:value variable="description"/></td>
            <td><webm:value variable="status"/></td>
        </tr>
        </webm:loop>
    </table>
</webm:invoke>
```

- To code loops over the pipeline and insert the names and values of the keys in the pipeline into the HTML page:

```
<webm:loop loopStruct="true">
    <webm:value variable="$key"/><br>
    <webm:value/><br>
</webm:loop>
```

- To insert the contents of the variable carrier into the HTML page or insert the string "UPS" if the carrier value is null or empty:

```
<webm:value variable="carrier" null="UPS" empty="UPS"/>
```

DSP Equivalents

The webMethods tags for JSP that Integration Server supports are similar to webMethods DSP tags. For information about webMethods DSP tags, including their arguments and options, see *Dynamic Server Pages and Output Templates Developer's Guide*. The following table lists the tags in the webMethods tag library and their DSP equivalents.

webMethods Tag for JSP	Equivalent DSP Tag
include	include
webm:case	case
webm:comment	comment
webm:else	else
webm:ifvar	ifvar
webm:invoke	invoke
webm:loop	loop
webm:loopsep	loopsep
webm:nl	nl
webm:onError	onerror
webm:onSuccess	None
webm:rename	rename
webm:scope	scope
webm:switch	switch
webm:sysvar	sysvar

webMethods Tag for JSP	Equivalent DSP Tag
webm:then	None. DSPs execute the block of code immediately following the ifvar tag.
webm:usePipeline	None. By default, the pipeline is in the scope of every service invocation.
webm:value	value