

GateIn Reference Guide



by the GateIn community , JBoss by Red Hat , and eXo Platform

edited by Scott Mumford (Red Hat), Thomas Heute (Red Hat), Luc Texier (Red Hat), and Christophe Laprun (Red Hat)

1. Web Services for Remote Portlets (WSRP)	1
1.1. Introduction	1
1.2. Level of support in GateIn 3.1	1
1.3. Deploying GateIn's WSRP services	2
1.3.1. Considerations to use WSRP when running GateIn on a non-default port or hostname	3
1.3.2. Considerations to use WSRP with SSL	3
1.4. Making a portlet remotable	3
1.5. Consuming GateIn's WSRP portlets from a remote Consumer	5
1.6. Consuming remote WSRP portlets in GateIn	5
1.6.1. Overview	5
1.6.2. Configuring a remote producer walk-through	5
1.6.3. Configuring access to remote producers via XML	10
1.6.4. Examples	12
1.7. Consumers maintenance	14
1.7.1. Modifying a currently held registration	14
1.7.2. Consumer operations	17
1.7.3. Erasing local registration data	18
1.8. Configuring GateIn's WSRP Producer	19
1.8.1. Overview	19
1.8.2. Default configuration	19
1.8.3. Registration configuration	19
1.8.4. WSRP validation mode	22

Web Services for Remote Portlets (WSRP)

1.1. Introduction

The Web Services for Remote Portlets specification defines a web service interface for accessing and interacting with interactive presentation-oriented web services. It has been produced through the efforts of the Web Services for Remote Portlets (WSRP) OASIS Technical Committee. It is based on the requirements gathered and on the concrete proposals made to the committee.

Scenarios that motivate WSRP functionality include:

- Content hosts, such as portal servers, providing Portlets as presentation-oriented web services that can be used by aggregation engines.
- Aggregating frameworks, including portal servers, consuming presentation-oriented web services offered by content providers and integrating them into the framework.

More information on WSRP can be found on the [official website for WSRP](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp) [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp]. We suggest reading the [primer](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp) [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp] for a good, albeit technical, overview of WSRP.

1.2. Level of support in GateIn 3.1

The WSRP Technical Committee defined [WSRP Use Profiles](http://www.oasis-open.org/committees/download.php/3073) [http://www.oasis-open.org/committees/download.php/3073] to help with WSRP interoperability. We will refer to terms defined in that document in this section.

GateIn provides a Simple level of support for our WSRP Producer except that out-of-band registration is not currently handled. We support in-band registration and persistent local state (which are defined at the Complex level).

On the Consumer side, GateIn provides a Medium level of support for WSRP, except that we only handle HTML markup (as GateIn itself doesn't handle other markup types). We do support explicit portlet cloning and we fully support the PortletManagement interface.

As far as caching goes, we have Level 1 Producer and Consumer. We support Cookie handling properly on the Consumer and our Producer requires initialization of cookies (as we have found that it improved interoperability with some consumers). We don't support custom window states or modes, as Portal doesn't either. We do, however, support CSS on both the Producer (though it's more a function of the portlets than inherent Producer capability) and Consumer.

While we provide a complete implementation of WSRP 1.0, we do need to go through the [Conformance statements](http://www.oasis-open.org/committees/download.php/6018) [http://www.oasis-open.org/committees/download.php/6018] and perform more interoperability testing (an area that needs to be better supported by the WSRP Technical Committee and Community at large).



Note

As of version 3.1 of GateIn, WSRP is only activated and supported when GateIn is deployed on JBoss Application Server.

1.3. Deploying GateIn's WSRP services

GateIn provides a complete support of WSRP 1.0 standard interfaces and offers both consumer and producer services. WSRP support is provided by the following files, assuming `$GATEIN_HOME` is where GateIn has been installed, `$WSRP_VERSION` (at the time of the writing, it was 1.1.0-GA) is the version of the WSRP component and `$PORTAL_VERSION` (at the time of the writing, it was 3.0.1-GA) is the current GateIn version:

- `$GATEIN_HOME/wsrp-admin-gui.war`, which contains the WSRP Configuration portlet with which you can configure consumers to access remote servers and how the WSRP producer is configured.
- `$GATEIN_HOME/wsrp-producer.war`, which contains the WSRP producer web application.
- `$GATEIN_HOME/lib/wsrp-common-$WSRP_VERSION.jar`, which contains common classes needed by the different WSRP libraries.
- `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar`, which contains the WSRP consumer.
- `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar`, which contains the API classes needed to integrate the WSRP component into portals.
- `$GATEIN_HOME/lib/wsrp-producer-lib-$WSRP_VERSION.jar`, which contains the classes needed by the WSRP producer.
- `$GATEIN_HOME/lib/wsrp-wsrp1-ws-$WSRP_VERSION.jar`, which contains the generated JAX-WS classes for WSRP version 1.
- `$GATEIN_HOME/lib/gatein.portal.component.wsrp-$PORTAL_VERSION.jar`, which contains the code to integrate the WSRP service into GateIn.

If you're not going to use WSRP in GateIn, you can remove `$GATEIN_HOME/lib/gatein.portal.component.wsrp-$PORTAL_VERSION.jar` from your GateIn distribution to easily deactivate WSRP support. Of course, if you want to trim your installation, you can also remove all the files mentioned above.

1.3.1. Considerations to use WSRP when running GateIn on a non-default port or hostname

JBoss WS (the web service stack that GateIn uses) should take care of the details of updating the port and host name used in WSDL. See the [JBoss WS user guide on that subject](http://community.jboss.org/wiki/JBossWS-UserGuide#Configuration) [http://community.jboss.org/wiki/JBossWS-UserGuide#Configuration] for more details.

Of course, if you have modified you have modified the host name and port on which your server runs, you will need to update the configuration for the consumer used to consume GateIn's 'self' producer. Please refer to the [Section 1.6, "Consuming remote WSRP portlets in GateIn"](#) to learn how to do so.

1.3.2. Considerations to use WSRP with SSL

It is possible to use WSRP over SSL for secure exchange of data. Please refer to the [instructions](#) [http://community.jboss.org/wiki/ConfiguringWSRPforuseoverSSL] on how to do so from [GateIn's wiki](#) [http://community.jboss.org/wiki/GateIn].

1.4. Making a portlet remotable

GateIn does **NOT**, by default, expose local portlets for consumption by remote WSRP consumers. In order to make a portlet remotely available, it must be made "remotable" by marking it as such in the associated `portlet.xml`. This is accomplished by using a specific `org.gatein.pc.remotable container-runtime-option`. Setting its value to `true` makes the portlet available for remote consumption, while setting its value to `false` will not publish it remotely. As specifying the remotable status for a portlet is optional, you do not need to do anything if you don't need your portlet to be available remotely.

In the following example, the "BasicPortlet" portlet is specified as being remotable.

Example 1.1.

```
<?xml version="1.0" standalone="yes"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
<portlet-app>
  <portlet>
    <portlet-name>BasicPortlet</portlet-name>

    ...
```

```
<container-runtime-option>
  <name>org.gatein.pc.remotable</name>
  <value>>true</value>
</container-runtime-option>
</portlet>
</portlet-app>
```

It is also possible to specify that all the portlets declared within a given portlet application to be remotable by default. This is done by specifying the `container-runtime-option` at the `portlet-app` element level. Individual portlets can override that value to not be remotely exposed. Let's look at an example:

Example 1.2.

```
<?xml version="1.0" standalone="yes"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
<portlet-app>

  <portlet>
    <portlet-name>RemotelyExposedPortlet</portlet-name>
    ...
  </portlet>
  <portlet>
    <portlet-name>NotRemotelyExposedPortlet</portlet-name>
    ...
    <container-runtime-option>
      <name>org.gatein.pc.remotable</name>
      <value>>false</value>
    </container-runtime-option>
  </portlet>

  <container-runtime-option>
    <name>org.gatein.pc.remotable</name>
    <value>>true</value>
  </container-runtime-option>
</portlet-app>
```


In the example above, we defined two portlets. The `org.gatein.pc.remotable container-runtime-option` being set to `true` at the `portlet-app` level, all portlets defined in this particular portlet application are exposed remotely by GateIn's WSRP producer. Note, however, that it is possible to override the default behavior: specifying a value for the `org.gatein.pc.remotable container-runtime-option` at the `portlet` level will take precedence over the default. In the example above, the `RemotelyExposedPortlet` inherits the `remotable` status defined at the `portlet-app` level since it does not specify a value for the `org.gatein.pc.remotable container-runtime-option`. The `NotRemotelyExposedPortlet`, however, overrides the default behavior and is not remotely exposed. Note that in the absence of a top-level `org.gatein.pc.remotable container-runtime-option` value set to `true`, portlets are NOT remotely exposed.

1.5. Consuming GateIn's WSRP portlets from a remote Consumer

WSRP Consumers vary a lot as far as how they are configured. Most of them require that you specify the URL for the Producer's WSDL definition. Please refer to your Consumer's documentation for specific instructions. For instructions on how to do so in GateIn, please refer to [Section 1.6, "Consuming remote WSRP portlets in GateIn"](#).

GateIn's Producer is automatically set up when you deploy a portal instance with the WSRP service. You can access the WSDL file at `http://{hostname}:{port}/portal-wsrp/MarkupService?wsdl`. The default hostname is `localhost` and the default port is `8080`.

1.6. Consuming remote WSRP portlets in GateIn

1.6.1. Overview

To be able to consume WSRP portlets exposed by a remote producer, GateIn's WSRP consumer needs to know how to access that remote producer. One can configure access to a remote producer using WSRP Producer descriptors. Alternatively, a portlet is provided to configure remote producers.

Once a remote producer has been configured, the portlets that it exposes are then available in the Application Registry to be added to categories and then to pages.

As a way to test the WSRP producer service and to check that the portlets that you want to expose remotely are correctly published via WSRP, a default consumer named `self`, that consumes the portlets exposed by GateIn's producer, has been configured.

1.6.2. Configuring a remote producer walk-through

Let's work through the steps of defining access to a remote producer so that its portlets can be consumed within GateIn. We will configure access to Oracle's public WSRP producer. We will first examine how to do so using the configuration portlet. We will then show how the same result can

be accomplished with a producer descriptor, though it is far easier to do so via the configuration portlet.

1.6.2.1. Using the configuration portlet

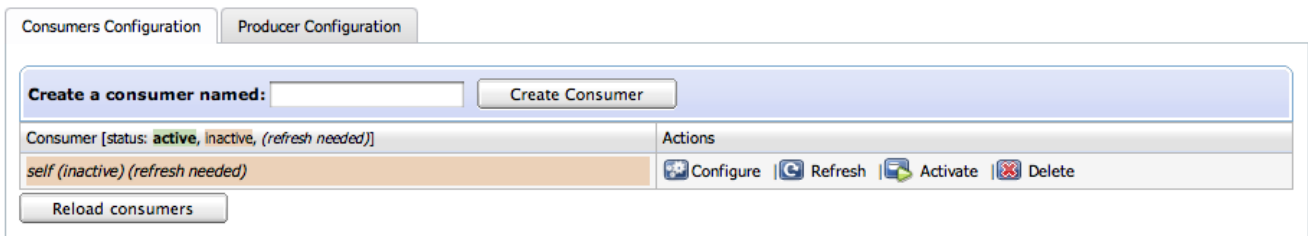
GateIn provides a portlet to configure access (among other functions) to remote WSRP Producers graphically. It isn't, however, installed by default, so the first thing we will need to do is to install the WSRP configuration portlet using the Application Registry.

Use the usual procedure to log in as a Portal administrator and go to the Application Registry. With the default install, you can just go to <http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2Fadministration%2Fregistry&username=root&password=gtn> [http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2Fadministration%2Fregistry&username=root&password=gtn]

Add the WSRP Configuration portlet to the Administration category. If you use the Import Applications functionality, the WSRP Configuration portlet will be automatically added to the Administration category.

Now that the portlet is added to a category, it can be added to a page and used. We recommend adding it to the same page as the Application Registry as operations relating to WSRP and adding portlets to categories are somewhat related as we will see. Go ahead and add the WSRP Configuration portlet to the page using the standard procedure.

If all went well, you should see something similar to this:



This screen presents all the configured producers associated with their status and possible actions on them. A Consumer can be active or inactive. Activating a Consumer means that it is ready to act as a portlet provider. Note also that a Consumer can be marked as requiring refresh meaning that the information held about it might not be up to date and refreshing it from the remote Producer might be a good idea. This can happen for several reasons: the service description for that remote Producer has not been fetched yet, the cached version has expired or modifications have been made to the configuration that could potentially invalidate it, thus requiring re-validation of the information.

Next, we create a new Consumer which we will call `oracle`. Type " `oracle`" in the "Create a consumer named:" field then click on "Create consumer":



You should now see a form allowing you to enter/modify the information about the Consumer. Set the cache expiration value to 300 seconds, leave the default timeout value for web services (WS) operations and enter the WSDL URL for the producer in the text field and press the "Refresh & Save" button:

Consumer 'oracle' configuration inactive (refresh needed)

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

This will retrieve the service description associated with the Producer which WSRP interface is described by the WSDL file found at the URL you just entered. In our case, querying the service description will allow us to learn that the Producer requires registration but didn't request any registration property:

✔ Refresh was successful.

Consumer 'oracle' configuration active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:


Current registration information:

Registration is indicated as required without registration properties.

Registration context: Handle:C:148.87.122.191:3fa5ac:1274c6b80c7:1de6

The Consumer for the `oracle` Producer should now be available as a portlet provider and be ready to be used.

Now, assuming that the producer required a value for an `email` registration property, GateIn's WSRP consumer would have informed you that you were missing some information:

 **Error: Refresh failed (probably because the registration information was not valid).**

Consumer 'self' configuration **inactive**


Producer id:


Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:		
Name	Description	Value
email	A valid contact email.	<input type="text"/>  Error: Missing value

 **Note**

At this point, there is no automated way to learn about which possible values (if any) are expected by the remote Producer. Sometimes, the possible values will be indicated in the registration property description but this is not always the case... Please refer to the specific Producer's documentation.

If you entered "example@example.com" as the value for the registration property and press "Save & Refresh" once more, you would have seen something similar to:

 **Refresh was successful.**

Consumer 'self' configuration **active**

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:		
Name	Description	Value
email	A valid contact email.	<input type="text" value="example@example.com"/>

Registration context: Handle:03b0a000c0a801327ec0c8c50ffa27db

1.6.2.2. Using XML

While we recommend you use the WSRP Configuration portlet to configure Consumers, we provide an alternative way to configure consumers by editing the XML file located at `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml`.

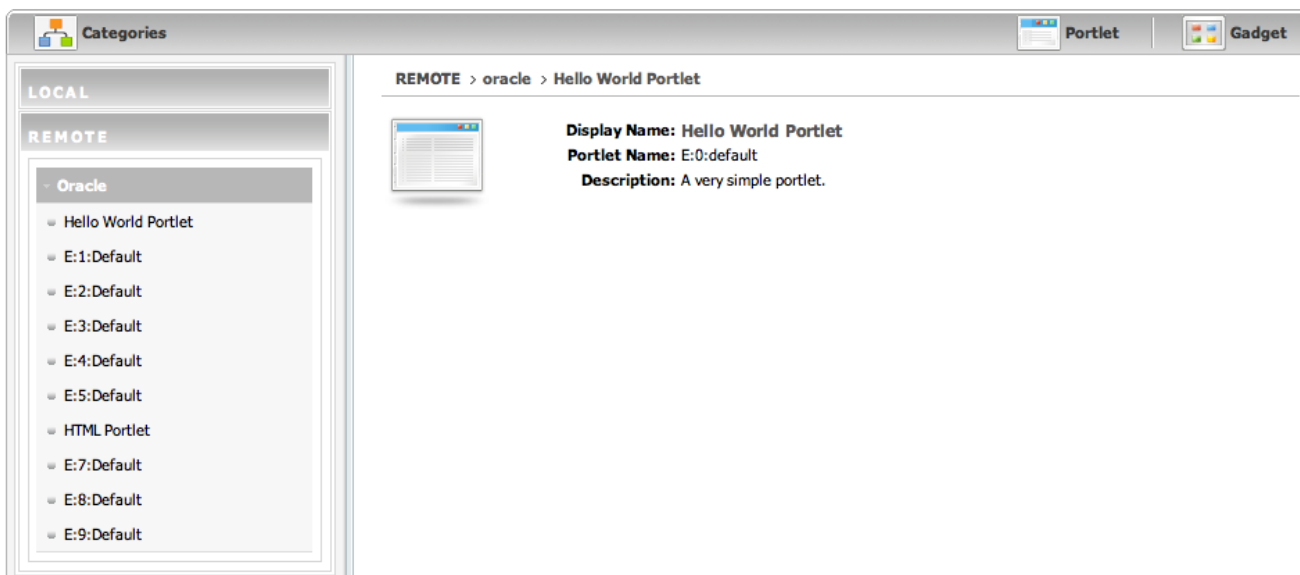
```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
  <deployment>
    <wsrp-producer id="self" expiration-cache="300" ws-timeout="30000">
      <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/MarkupService?wsdl</endpoint-
wsdl-url>
      <registration-data>
        <property>
          <name>email</name>
          <lang>en</lang>
          <value>example@example.com</value>
        </property>
      </registration-data>
    </wsrp-producer>
  </deployment>
  <deployment>
    <wsrp-producer id="oracle" expiration-cache="300">
      <endpoint-wsdl-url>http://portalstandards.oracle.com/portletapp/portlets?WSDL</endpoint-
wsdl-url>
      <registration-data/>
    </wsrp-producer>
  </deployment>
</deployments>
```

The file as shown above specifies access to two producers: `self`, which consumes GateIn's own WSRP producer albeit in a version that assumes that the producer requires a value for an `email` registration property, and `oracle`, which consumes Oracle's public producer, both in configurations as shown in the walk-through above.

We will look at the details of the meaning of elements later on.

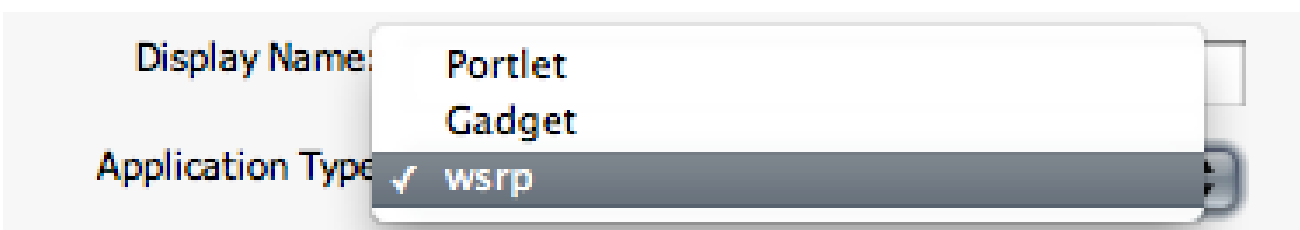
1.6.2.3. Configuring access to a remote portlet

If we go back to the Application Registry and examine the available portlets by clicking on the Portlet link, you will now be able to see the remote portlets if you click on the REMOTE tab in the left column:



These portlets are, of course, available to be used such as regular portlets: they can be used in categories and added to pages. If you use the Import Applications functionality, they will also be automatically imported in categories based on the keywords they define.

More specifically, if you want to add a WSRP portlet to a category, you can access these portlets by selecting `wsrp` in the Application Type drop-down menu:



1.6.3. Configuring access to remote producers via XML

While we recommend you use the WSRP Configuration portlet to configure Consumers, we provide an alternative way to configure consumers by editing the XML file located at `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml`.



Note

An XML Schema defining which elements are available to configure Consumers via XML can be found in `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar/xsd/gatein_wsrp_consumer_1_0.xsd`



Note

It is important to note how the XML consumers configuration file is processed. It is read the first time the WSRP service starts and the associated information is then put under control of JCR (Java Content Repository). Subsequent launches of the WSRP service will use the JCR-stored information for all producers already known to GateIn. More specifically, `wsrp-consumers-config.xml` file is scanned for producer identifiers. Any identifier that is already known will be bypassed and the JCR information associated with this remote producer will be used. The information defined at the XML level is only processed for producer definition for which no information is already present in JCR. Therefore, if you wish to delete a producer configuration, you need to delete the associated information in the database (this can be accomplished using the configuration portlet as we saw in [Section 1.6.2.1, "Using the configuration portlet"](#)) AND remove the associated information in `wsrp-consumers-config.xml` (if such information exists) as the producer will be re-created the next time the WSRP is launched if that information is not removed.

1.6.3.1. Required configuration information

Let's now look at which information needs to be provided to configure access to a remote producer.

First, we need to provide an identifier for the producer we are configuring so that we can refer to it afterwards. This is accomplished via the mandatory `id` attribute of the `<wsrp-producer>` element.

GateIn also needs to learn about the remote producer's endpoints to be able to connect to the remote web services and perform WSRP invocations. This is accomplished by specifying the URL for the WSDL description for the remote WSRP service, using the `<endpoint-wsdl-url>` element.

Both the `id` attribute and `<endpoint-wsdl-url>` elements are required for a functional remote producer configuration.

1.6.3.2. Optional configuration

It is also possible to provide additional configuration, which, in some cases, might be important to establish a proper connection to the remote producer.

One such optional configuration concerns caching. To prevent useless roundtrips between the local consumer and the remote producer, it is possible to cache some of the information sent

by the producer (such as the list of offered portlets) for a given duration. The rate at which the information is refreshed is defined by the `expiration-cache` attribute of the `<wsrp-producer>` element which specifies the refreshing period in seconds. For example, providing a value of 120 for `expiration-cache` means that the producer information will not be refreshed for 2 minutes after it has been somehow accessed. If no value is provided, GateIn will always access the remote producer regardless of whether the remote information has changed or not. Since, in most instances, the information provided by the producer does not change often, we recommend that you use this caching facility to minimize bandwidth usage.

It is also possible to define a timeout after which WS operations are considered as failed. This is helpful to avoid blocking the WSRP service, waiting forever on the service that doesn't answer. Use the `ws-timeout` attribute of the `<wsrp-producer>` element to specify how many milliseconds the WSRP service will wait for a response from the remote producer before timing out and giving up.

Additionally, some producers require consumers to register with them before authorizing them to access their offered portlets. If you know that information beforehand, you can provide the required registration information in the producer configuration so that the consumer can register with the remote producer when required.



Note

At this time, though, only simple String properties are supported and it is not possible to configure complex registration data. This should, however, be sufficient for most cases.

Registration configuration is done via the `<registration-data>` element. Since GateIn can generate the mandatory information for you, if the remote producer does not require any registration properties, you only need to provide an empty `<registration-data>` element. Values for the registration properties required by the remote producer can be provided via `<property>` elements. See the example below for more details. Additionally, you can override the default consumer name automatically provided by GateIn via the `<consumer-name>` element. If you choose to provide a consumer name, please remember that this should uniquely identify your consumer.

1.6.4. Examples

Here is the configuration of the `self` producer as found in `default-wsrp.xml` with a cache expiring every five minutes and with a 30 second timeout for web service operations.

Example 1.3.

```
<?xml version='1.0' encoding='UTF-8' ?>
```



```

<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
  <deployment>
    <wsrp-producer id="self" expiration-cache="300" ws-timeout="30000">
      <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/MarkupService?wsdl</endpoint-
wsdl-url>
      <registration-data/>
    </wsrp-producer>
  </deployment>
</deployments>

```

Here is an example of a WSRP descriptor with registration data and cache expiring every minute:

Example 1.4.

```

<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
<deployments>
  <deployment>
    <wsrp-producer id="AnotherProducer" expiration-cache="60">
      <endpoint-wsdl-url>http://example.com/producer/producer?WSDL</endpoint-wsdl-url>
      <registration-data>
        <property>
          <name>property name</name>
          <lang>en</lang>
          <value>property value</value>
        </property>
      </registration-data>
    </wsrp-producer>
  </deployment>
</deployments>

```

1.7. Consumers maintenance

1.7.1. Modifying a currently held registration

1.7.1.1. Registration modification for service upgrade

Producers often offer several levels of service depending on consumers' subscription levels (for example). This is implemented at the WSRP level with the registration concept: producers can assert which level of service to provide to consumers based on the values of given registration properties.

There might also be cases where you just want to update the registration information because it has changed. For example, the producer required you to provide a valid email and the previously email address is not valid anymore and needs to be updated.

It is therefore sometimes necessary to modify the registration that concretizes the service agreement between a consumer and a producer. Let's take the example of the producer requiring an email we configured in [Section 1.6.2.1, "Using the configuration portlet"](#). If you recall, the producer was requiring registration and required a value to be provided for the `email` property.

Suppose now that we would like to update the email address that we provided to the remote producer. We will need to tell the producer that our registration data has been modified. Let's see how to do this. Assuming you have configured access to the producer as previously described, please go to the configuration screen for the `self` producer and modify the value of `email` to `foo@example.com` instead of `example@example.com`:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Now click on "Update properties" to save the change. A "Modify registration" button should now appear to let you send this new data to the remote producer:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Click on this new button and, if everything went well and your updated registration has been accepted by the remote producer, you should see something similar to:

- ✔ Successfully modified registration!
- ✔ Refresh was successful.

Consumer 'self' configuration active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Registration context: Handle:07d57d29c0a801325a0da57a96c12a32

1.7.1.2. Registration modification on producer error

It can also happen that a producer administrator decided to change its requirement for registered consumers. In this case, invoking operations on the producer will fail with an `OperationFailedFault`. GateIn will attempt to help you in this situation. Let's walk through an example using the `self` producer. Let's assume that registration is requiring a valid value for an `email` registration property (as we have seen so far). If you go to the configuration screen for this producer, you should see:

Consumer 'self' configuration active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Registration context: Handle:07d57d29c0a801325a0da57a96c12a32

Now suppose that the administrator of the producer now additionally requires a value to be provided for a `name` registration property. We will actually see how to do perform this operation in

GateIn when we examine how to configure GateIn's producer in [Section 1.8, "Configuring GateIn's WSRP Producer"](#). Operations with this producer will now fail. If you suspect that a registration modification is required, you should go to the configuration screen for this remote producer and refresh the information held by the consumer by pressing "Refresh & Save":

Error: Either local or remote information has been changed, you should modify your registration with the remote producer. The new local information will be saved but your current registration data will be used until you successfully modify the registration with the producer.

Consumer 'self' configuration inactive (refresh needed)

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>

Expected registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>
name	A contact name.	<input type="text" value=""/> Error: Missing value

Registration context: Handle:07d57d29c0a801325a0da57a96c12a32

As you can see, the configuration screen now shows the currently held registration information and the expected information from the producer. Enter a value for the `name` property and then click on "Modify registration". If all went well and the producer accepted your new registration data, you should see something similar to:

- ✔ Successfully modified registration!
- ✔ Refresh was successful.

Consumer 'self' configuration active

Producer id:

Cache expiration: (seconds before expiration)

Timeout for WS operations: (milliseconds before timeout)

Producer WSDL URL:

Registration information:

Current registration information:

Name	Description	Value
email	A valid email.	<input type="text" value="foo@example.com"/>
name	A contact name.	<input type="text" value="Foo Bar"/>

Registration context: Handle:07d57d29c0a801325a0da57a96c12a32



Note

As of WSRP 1, it is rather difficult to ascertain for sure what caused an `OperationFailedFault` as it is the generic exception returned by producers if something didn't quite happen as expected during a method invocation. This means that `OperationFailedFault` can be caused by several different reasons, one of them being a request to modify the registration data. Please take a look at the log files to see if you can gather more information as to what happened. WSRP 2 introduces an exception that is specific to a request to modify registrations thus reducing the ambiguity that currently exists.

1.7.2. Consumer operations

Several operations are available from the consumer list view of the WSRP configuration portlet:

Create a consumer named:

Consumer [status: active , inactive , (refresh needed)]	Actions
self (active)	<input type="button" value="Configure"/> <input type="button" value="Refresh"/> <input type="button" value="Deactivate"/> <input type="button" value="Deregister"/> <input type="button" value="Delete"/>

The available operations are:

- Configure: displays the consumer details and allows user to edit them

- Refresh: forces the consumer to retrieve the service description from the remote producer to refresh the local information (offered portlets, registration information, etc.)
- Activate/Deactivate: activates/deactivates a consumer, governing whether it will be available to provide portlets and receive portlet invocations
- Register/Deregister: registers/deregisters a consumer based on whether registration is required and/or acquired
- Delete: destroys the consumer, after deregistering it if it was registered

1.7.3. Erasing local registration data


There are rare cases where it might be required to erase the local information without being able to deregister first. This is the case when a consumer is registered with a producer that has been modified by its administrator to not require registration anymore. If that ever was to happen (most likely, it won't), you can erase the local registration information from the consumer so that it can resume interacting with the remote producer. To do so, click on "Erase local registration" button next to the registration context information on the consumer configuration screen:

Registration context:

Handle:07d57d29c0a801325a0da57a96c12a32

Erase local registration

Warning: This operation is dangerous as it can result in inability to interact with the remote producer if invoked when not required. A warning screen will be displayed to give you a chance to change your mind:

 **Delete local registration for 'self' consumer?**

Warning: You are about to delete the local registration information for the 'self' consumer! This is only needed if this consumer had previously registered with the remote producer and this producer has been modified to not require registration anymore. Only erase local registration information if you experience errors with the producer due to this particular situation. Erasing local registration when not required might lead to inability to work with this producer anymore.

Are you sure you want to proceed?

Erase local registration Cancel

1.8. Configuring GateIn's WSRP Producer

1.8.1. Overview

You can configure the behavior of Portal's WSRP Producer by using the WSRP administration interface, which is the preferred way, or by editing the `$GATEIN_HOME/wsrp-producer.war/WEB-INF/conf/producer/config.xml` file. Several aspects can be modified with respects to whether registration is required for consumers to access the Producer's services. An XML Schema for the configuration format is available at `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar/xsd/gatein_wsrp_producer_1_0.xsd`.

1.8.2. Default configuration

The default producer configuration is to require that consumers register with it before providing access its services but does not require any specific registration properties (apart from what is mandated by the WSRP standard). It does, however, require consumers to be registered before sending them a full service description. This means that our WSRP producer will not provide the list of offered portlets and other capabilities to unregistered consumers. The producer also uses the default `RegistrationPolicy` paired with the default `RegistrationPropertyValidator`. We will look into property validators in greater detail later in [Section 1.8.3, "Registration configuration"](#). Suffice to say for now that this allows users to customize how Portal's WSRP Producer decides whether a given registration property is valid or not.

GateIn provides a web interface to configure the producer's behavior. You can access it by clicking on the "Producer Configuration" tab of the "WSRP" page of the "admin" portal. Here's what you should see with the default configuration:

Access to full service description requires consumers to be registered.

Use strict WSRP compliance.

Requires registration. Modifying this information will trigger invalidation of consumer registrations.

Registration policy class name:

Registration property validator class name:

Registration properties

No specified required registration properties.

As would be expected, you can specify whether or not the producer will send the full service description to unregistered consumers, and, if it requires registration, which `RegistrationPolicy` to use (and, if needed, which `RegistrationPropertyValidator`), along with required registration property description for which consumers must provide acceptable values to successfully register.

1.8.3. Registration configuration

In order to require consumers to register with Portal's producer before interacting with it, you need to configure Portal's behavior with respect to registration. Registration is optional, as are registration properties. The producer can require registration without requiring consumers to pass

any registration properties as is the case in the default configuration. Let's configure our producer starting with a blank state:

- Access to full service description requires consumers to be registered.
- Use strict WSRP compliance.
- Requires registration. Modifying this information will trigger invalidation of consumer registrations.

We will allow unregistered consumers to see the list of offered portlets so we leave the first checkbox ("Access to full service description requires consumers to be registered.") unchecked. We will, however, specify that consumers will need to be registered to be able to interact with our producer. Check the second checkbox ("Requires registration. Modifying this information will trigger invalidation of consumer registrations."). The screen should now refresh and display:

Consumers Configuration

Producer Configuration

- Access to full service description requires consumers to be registered.
- Use strict WSRP compliance.
- Requires registration. Modifying this information will trigger invalidation of consumer registrations.

Registration policy class name:

Registration property validator class name:

Registration properties

No specified required registration properties.

You can specify the fully-qualified name for your `RegistrationPolicy` and `RegistrationPropertyValidator` there. We will keep the default value. See [Section 1.8.3.1, "Customization of Registration handling behavior"](#) for more details. Let's add, however, a registration property called `email`. Click "Add property" and enter the appropriate information in the fields, providing a description for the registration property that can be used by consumers to figure out its purpose:

- Access to full service description requires consumers to be registered.
- Use strict WSRP compliance.
- Requires registration. Modifying this information will trigger invalidation of consumer registrations.

Registration policy class name:

Registration property validator class name:

Registration properties

Name	Type	Label	Hint	Action
<input type="text" value="email"/>	<input type="text" value="xsd:string"/>	<input type="text" value="A valid contact email."/>	<input type="text" value="A valid contact email."/>	<input type="button" value="Remove"/>

Press "Save" to record your modifications.

**Note**

At this time, only String (xsd:string) properties are supported. If your application requires more complex properties, please let us know.

**Note**

If consumers are already registered with the producer, modifying the configuration of required registration information will trigger the invalidation of held registrations, requiring consumers to modify their registration before being able to access the producer again. We saw the consumer side of that process in [Section 1.7.1.2, "Registration modification on producer error"](#).

1.8.3.1. Customization of Registration handling behavior

Registration handling behavior can be customized by users to suit their Producer needs. This is accomplished by providing an implementation of the `RegistrationPolicy` interface. This interface defines methods that are called by Portal's Registration service so that decisions can be made appropriately. A default registration policy that provides basic behavior is provided and should be enough for most user needs.

While the default registration policy provides default behavior for most registration-related aspects, there is still one aspect that requires configuration: whether a given value for a registration property is acceptable by the WSRP Producer. This is accomplished by plugging a `RegistrationPropertyValidator` in the default registration policy. This allows users to define their own validation mechanism.

Please refer to the Javadoc™ for `org.jboss.portal.registration.RegistrationPolicy` and `org.jboss.portal.registration.policies.RegistrationPropertyValidator` for more details on what is expected of each method.

Defining a registration policy is required for the producer to be correctly configured. This is accomplished by specifying the qualified class name of the registration policy. Since we anticipate that most users will use the default registration policy, it is possible to provide the class name of your custom property validator instead to customize the default registration policy behavior. Note that property validators are only used by the default policy.

**Note**

Since the policy or the validator are defined via their class name and dynamically loaded, it is important that you make sure that the identified class is available to the application server. One way to accomplish that is to deploy your policy implementation as JAR file in your AS instance deploy directory. Note also that,

since both policies and validators are dynamically instantiated, they must provide a default, no-argument constructor.

1.8.4. WSRP validation mode

The lack of conformance kit and the wording of the WSRP specification leaves room for differing interpretations, resulting in interoperability issues. It is therefore possible to encounter issues when using consumers from different vendors. We have experienced such issues and have introduced a way to relax the validation that our WSRP producer performs on the data provided by consumers to help with interoperability by accepting data that would normally be invalid. Note that we only relax our validation algorithm on aspects of the specification that are deemed harmless such as invalid language codes.

By default, the WSRP producer is configured in strict mode. If you experience issues with a given consumer, you might want to try to relax the validation mode. This is accomplished by unchecking the "Use strict WSRP compliance." checkbox on the Producer configuration screen.