webMethods

This topic explains how to configure and apply the webMethods tool, which allows you to publish and send BrokerEvent objects to Software AG webMethods Broker, as well as subscribe to events and invoke native webMethods IS (Integration Server) services. Sections include:

Working with Broker

The configuration steps vary according to the execution mode you want to use (publish, subscribe, request/reply).

Adding Required Jar Files to the Classpath

The following jar files always need to be added to the SOAtest classpath:

- wmbrokerclient.jar
- g11nutils.jar

The jar files can be found under [webmethods install dir]/Broker/lib/. For more details, please refer to webMethods Broker Client Java API Programmer's Guide> Getting Started> Using the webMethods Broker Java API.

In some cases, enttoolkit.jar and mail.jar are also required.

To add these jar files to SOAtest classpath, complete the following:

1. Choose **Parasoft> Preferences**.
2. Open the **Parasoft> System Properties** page.
3. Click the **Add JARS** button and choose and select the necessary JAR files to be added.

Using Publish Execution Mode

To configure the webMethods tool to publish BrokerEvent objects:

1. Open up the tool configuration panel.
2. Open the **Tool Settings** tab.
3. Specify tool settings as follows:
    a. From **webMethods,** select **Broker**.
    b. From **Execution mode,** select **Publish.**
    c. In the **Host, Broker Name, Client Group,** and **Client ID** fields, specify the settings for connecting to Broker.
    d. In the **AppName** field, enter any name that you want to use to identify the application (SOAtest).
    e. Click **Refresh.** The **Event Type** menu will then be populated with the available event types on the broker.
    f. (Optional) In **Subscription Filter,** specify a filter string in order to retrieve only the events with desired field values. For example, you may use expressions such as
    `my_Strin_field="some value" and my_int_value < 5`
    For more details on the syntax of filter strings, see the WebMethods Broker Client Java API Programmer's Guide, "Using Event Filters" section.
    g. In **Event Type,** select the event type to which you want to publish.
    h. In the **Timeout** area, customize the timeout period and timeout conditions if desired.
        - If the **Pass the test only if timeout occurred** option is selected, then the tool will fail if a Broker Event is received at the specified event type. This is typically useful for subscribing to error event types and ensuring that no error event types are published as part of a test scenario.
4. Open the **Inputs** tab.
5. Use the available controls to specify what BrokerEvent object you want published.
    - The available options (Form XML, Literal XML, Scripted, Form Input) and the related controls are the same as the ones available for the SOAP Client tool and other mes-saging tools. See Message Tool and Responder Options for details.
    - If you are using Form XML, Literal XML or Form Input and want to generate a default input template, click **Refresh.**
    - In scripted mode, you can directly manipulate the object (e.g, if you want to set the field values by scripting using the webMethods API instead of the GUI or XML representations). See Specifying Scripted Inputs and Extensibility and Scripting Basics for details.
    - For more information about BrokerEvent objects, please refer to the COM.activesw.api.client.BrokerEvent documentation in the webMethods Broker Client Java API Programmer's Guide.

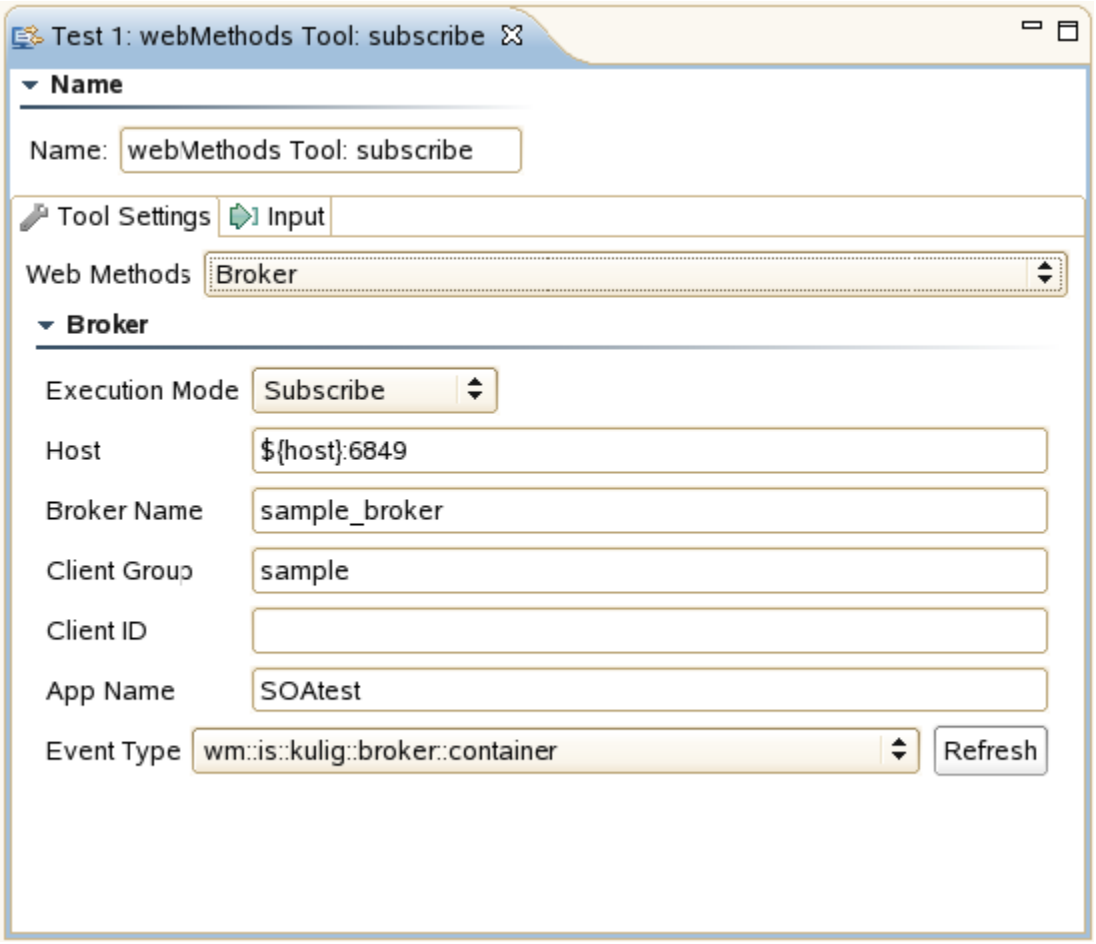
During test execution, SOAtest publishes the event by invoking the broker client Java API and providing it with a BrokerEventobject. Any applications that are subscribed to that event type will receive the event that SOAtest published.

To view an XML representation of the events that are published, use the Traffic Viewer tool, which is automatically chained to the webMethods tool. Note that even though actual BrokerEvent objects are sent and received, the viewer uses an XML representation. This makes it easier to read and allows you to chain tools that validate or process the content (e.g., the Diff tool or XML Assertor tool).

Using Subscribe Execution Mode

To configure the webMethods tool to subscribe to BrokerEvent objects:

1. Double-click the **webMethods** Test Case Explorer node to open up the tool configuration panel.
2. Open the **Tool Settings** tab.
3. Specify tool settings as follows:
    a. From **webMethods,** select **Broker.**
    b. From **Execution mode,** select **Subscribe.**
    c. In the **Host, Broker Name, Client Group,** and **Client ID** fields, specify the settings for connecting to Broker.
    d. In the **AppName** field, enter any name that you want to use to identify the application (SOAtest).
    e. Click **Refresh.** The **Event Type** menu will then be populated with the available event types on the broker.
    f. In **Event Type,** select the event type to which you want to subscribe.



During test execution, SOAtest starts listening for events that are published to the specified event type.

To view an XML representation of the events that are received, use the Traffic Viewer tool, which is automatically chained to the webMethods tool. Note that even though actual BrokerEvent event objects are sent and received, the viewer uses an XML representation. This makes it easier to read and allows you to chain tools that validate or process the content (e.g., the Diff tool or XML Assertor tool).

Using Request/Reply Execution Mode

To configure the webMethods tool to send a BrokerEvent object and wait for a reply before sending another:

1. Double-click the **webMethods** Test Case Explorer node to open up the tool configuration panel.
2. Open the **Tool Settings** tab.
3. Specify tool settings as follows:
    a. From **webMethods,** select **Broker**.
    b. From **Execution mode,** select **Request/Reply**.
    c. In the **Host, Broker Name, Client Group,** and **Client ID** fields, specify the settings for connecting to Broker.
    d. In the **AppName** field, enter any name that you want to use to identify the application (SOAtest).
    e. Click **Refresh.** The **Event Type** menu will then be populated with the available event types on the broker.
    f. In **Event Type,** select the event type to which you want to send.
4. Open the **Inputs** tab.
5. Use the available controls to specify what BrokerEvent object you want sent.

- The available options (Form XML, Literal XML, Scripted, Form Input) and the related controls are the same as the ones available for the SOAP Client tool and other messaging tools. See Message Tool and Responder Options for details.
- If you are using Form XML, Literal XML or Form Input and want to generate a default input template, click **Refresh**.
- In scripted mode, you can directly manipulate the object (e.g, if you want to set the field values by scripting using the webMethods API instead of the GUI or XML representations). See Specifying Scripted Inputs and Extensibility and Scripting Basics for details.
- For more information about BrokerEvent objects, please refer to the COM.activesw.api.client.BrokerEvent documentation in the webMethods Broker Client Java API Programmer's Guide.

During test execution, SOAtest sends a request by invoking the broker client Java API and providing it with a broker event object, then it waits for a reply.

To view an XML representation of the request and reply events, use the Traffic Viewer tool, which is automatically chained to the webMethods tool. Note that even though actual BrokerEvent objects are sent and received, the viewer uses an XML representation. This makes it easier to read and allows you to chain tools that validate or process the content (e.g., the Diff tool or XML Assertor tool).

Specifying Scripted Inputs

When publishing to a Broker or doing a request/reply using scripted input, two arguments will be passed into your custom method. The first argument is a BrokerEvent object for the type name you specified in the tool settings. You need to populate this object with the desired input values (see the webMethods Broker Client Java API Programmer's Guide for details on how to accomplish this) then return it. The second argument is a test tool com.parasoft.api.Context object.

Here is a sample Jython script:

```
from com.parasoft.api import *
from COM.activesw.api.client import *

def configureBrokerEvent(event, context):
    event.setUCStringField("s", "something")
    event.setIntegerField("i", 1)
    event.setUCStringSeqField("sl", 0, BrokerEvent.ENTIRE_SEQUENCE, BrokerEvent.AUTO_SIZE, ["one", "two",
"three"])
    return event
```

Working with Integration Server

Adding Required Jar Files to the Classpath

The following jar file needs to be added to the SOAtest classpath:

- wm-isclient.jar

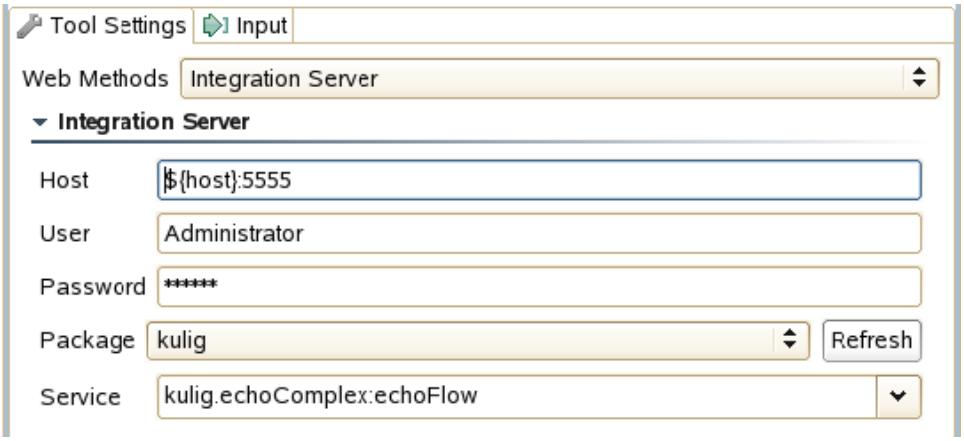
It can be found under [webmethods install dir]/common/lib/wm-isclient.jar. To add this jar file to SOAtest classpath, complete the following:

1. Choose **Parasoft> Preferences**.
2. Open the **Parasoft> System Properties** page.
3. Click the **Add JARS** button and choose and select the necessary JAR file to be added.

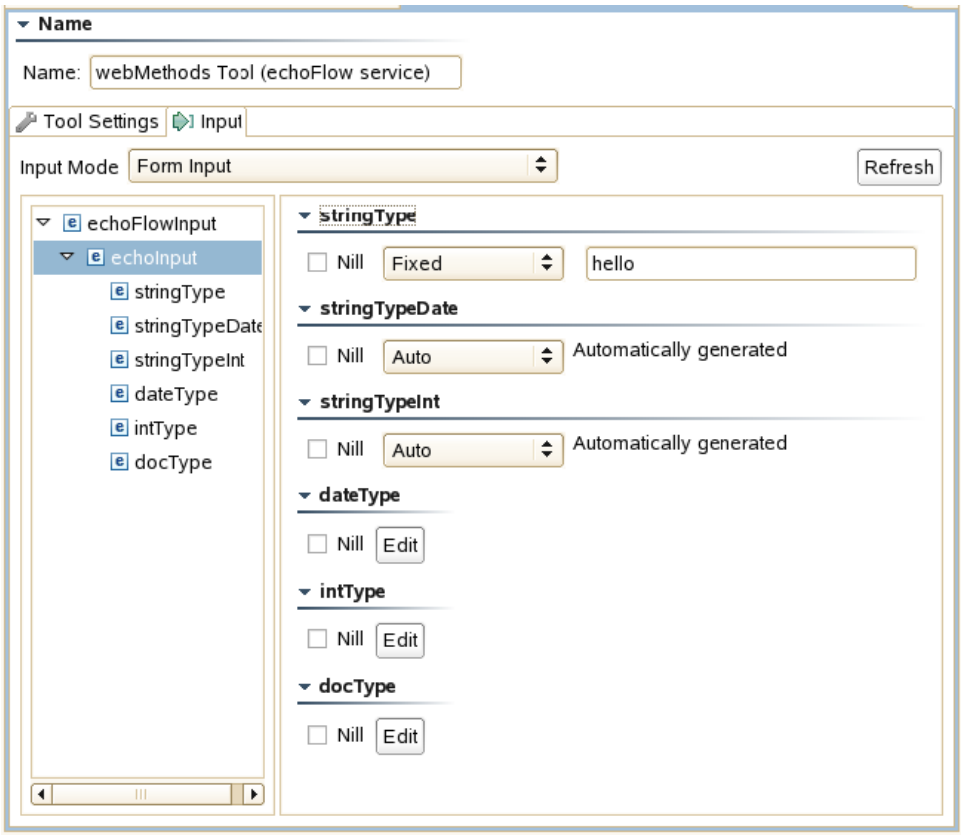
Using SOAtest with Integration Server

To configure SOAtest to invoke webMethods IS (Integration Server) services and receive responses:

1. Double-click the **webMethods** Test Case Explorer node to open up the tool configuration panel.
2. Open the **Tool Settings** tab.
3. Specify tool settings as follows:
    a. From **webMethods,** select **Integration Server.**
    b. In the **Host, User,** and **Password** fields, specify the settings for connecting to Integration Server.
    c. Click **Refresh**. The **Package** menu will then be populated with the packages available on Integration Server.
    d. In **Package,** choose the package that contains the service that you want to invoke.
    e. In **Service,** select the service that you want to invoke.

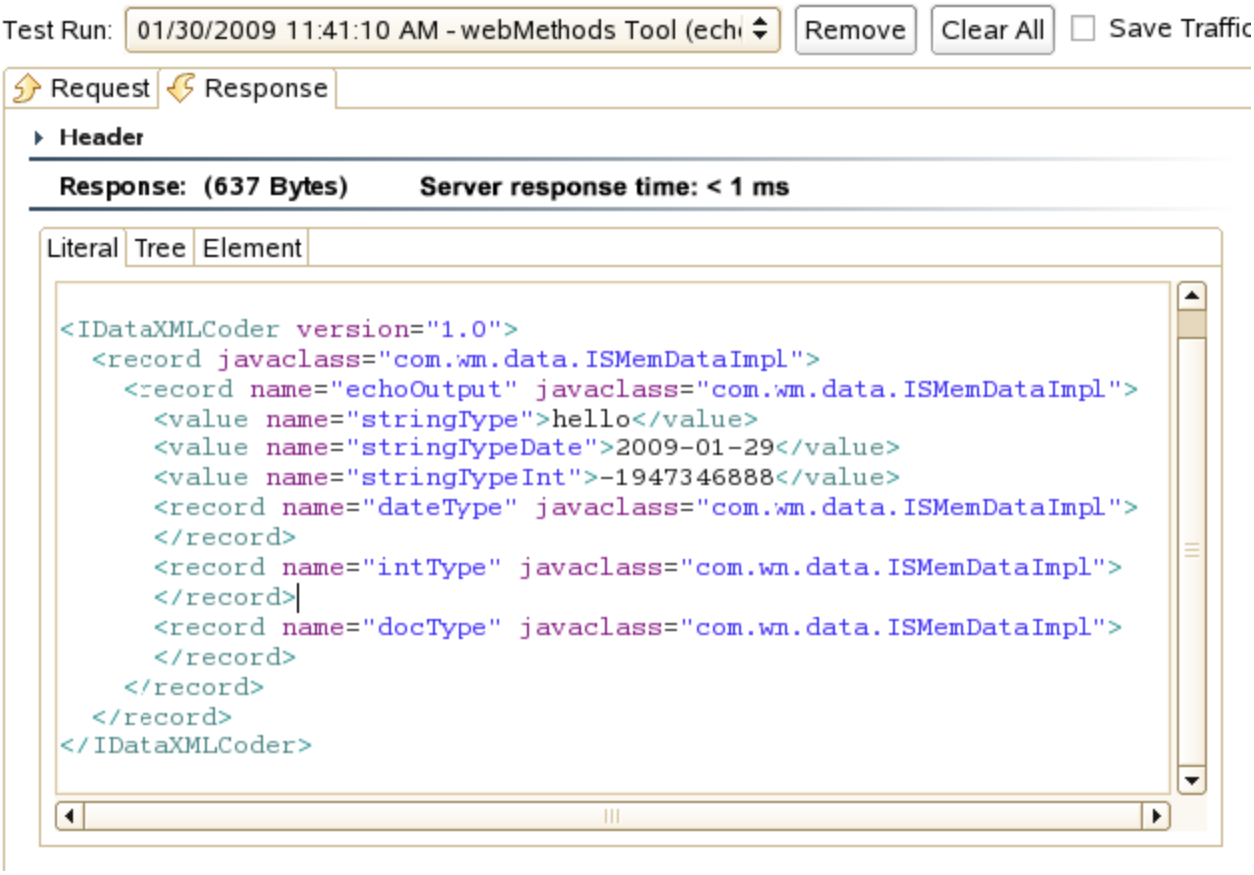


4. Open the **Input** tab.
5. Use the available controls to specify what iData object will be used to invoke that service.
    - The available options (Form XML, Literal XML, Scripted, Form Input) and the related controls are the same as the ones available for the SOAP Client tool and other messaging tools. See Message Tool and Responder Options for details.
    - If you are using Form XML, Literal XML or Form Input and want to generate a default input template, click **Refresh**.
    - In scripted mode, you can directly manipulate the object (e.g, if you want to set the field values by scripting using the webMethods API instead of the GUI or XML representations). See Specifying Scripted Inputs and Extensibility and Scripting Basics for details.
    - For more information about iData objects, please refer to the com.wm.data.iData documentation in the webMethods Integration Server Developer Guide.



During test execution, SOAtest sends iData objects to invoke Integration Server services and receives responses.

To view an XML representation of the objects that are sent and received, use the Traffic Viewer tool, which is automatically chained to the webMethods tool. This XML representation uses the iData XML coder format. Note that even though actual iData objects are sent and received, the viewer uses an XML representation. This makes it easier to read and allows you to chain tools that validate or process the content (e.g., the Diff tool or XML Assertor tool).



Specifying Scripted Inputs

When using an Integration Server service with scripted input, your custom method will be passed two arguments. The first is an IData object that you need to populate with the desired data and then return. The second argument is a com.parasoft.api.Context object.

Here is a sample Jython script:

```
from com.wm.app.b2b.client import *
from com.wm.data import *

def buildIData(idata, context):
    IDataUtil.put(idata.getCursor(), "in1", "12")
        IDataUtil.put(idata.getCursor(), "in2", "13")
        return idata
```

Accessing the Session ID

The session ID is automatically saved to the `WebMethodsSessionId` context variable. It can be accessed from an Extension tool, script, or custom tool using `com.parasoft.api.Context.get(String key)`, where `String key` is `"WebMethodsSessionId"`.

For example, the following Extension tool (using a Groovy script) accesses the session id that is stored in the `sessionId` variable,:

```
import com.parasoft.api.*
boolean useSessionId(input, context) {
    sessionId = context.get("WebMethodsSessionId")
        Application.showMessage("sessionId = " + sessionId)
    // Put code here to do something with the session id
        return true
}
```