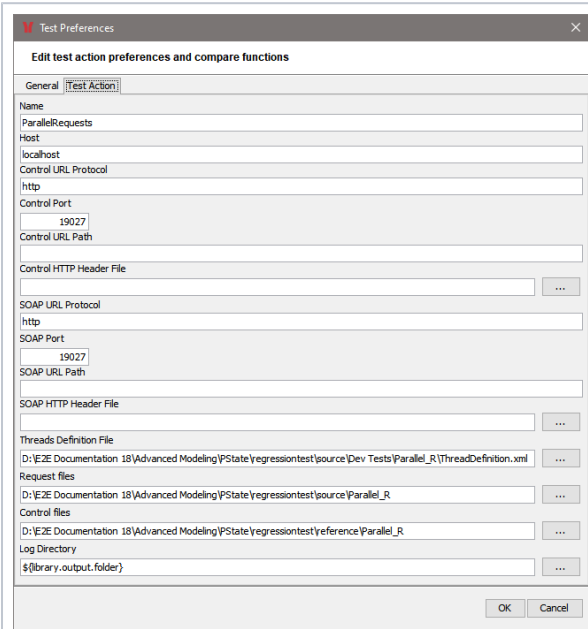


# Adding a ParallelRequest Test

With **ParallelRequest** tests you can run multiple tests in parallel. These tests can use up to two BRIDGE services.



Fill in the fields according to the description n below.

On this Page:

- [Defining Threads](#)
- [Working With a Variable Store](#)

Related Pages:

- [Options: Build Bridge Repository](#)
- [Adding a Build Test](#)
- [Adding a Deploy Test](#)
- [Adding a Start/Stop Test](#)
- [Adding a Wait Test](#)
- [Adding a ParallelRequest Test](#)
- [Adding a ServiceCommand Test](#)

Field	Description	Example
Name	Specify the name of the test.	
Host	Specify the host, the service has been deployed to.	localhost
Definitions Service 1 (Control)		
Specify the definitions for the service you want to use with ParallelRequests.		
Control URL Protocol	Specify the protocol.	HTTP or HTTPS
Control Port	Specify the port of the recipient, e.g. Bridge port or control port of the service you want to send a requests to.	8080 or 29027
Control URL Path	Specify the URL path of the request target, e.g. the Bridge API.	/bridge/rest /services/xuml /PurchaseOrderExample
Control HTTP Header File	Specify the path to a file that contains URL headers if necessary.	
Definitions Service 2 (SOAP)		
Specify the definitions for a second service you want to use with ParallelRequests.		
SOAP URL Protocol	Specify the protocol.	
SOAP Port	Specify the port of the recipient, e.g. Bridge port or control port of the service you want to send a requests to.	
SOAP URL Path	Specify the URL path of the request target.	
SOAP HTTP Header File	Specify the path to a file that contains URL headers if necessary.	

Threads Definitions		
<b>Threads Definition File</b>	Specify a path to an XML file that contains the thread definitions. For more details on the contents of this file, refer to <a href="#">Defining Threads</a> below.	Default name is ThreadDefinition.xml. This name may be changed.
<b>Request Files</b>	Specify a path to a folder that contains additional request files if necessary. For e.g. POST requests that would be the request body.	
<b>Control Files</b>	Change here the folder, where the reference data is stored.	
<b>Log Directory</b>	Change here the folder, where the test action results get stored.	

## Defining Threads

The actual parallel threads cannot be defined via the Analyzer but you have to define them in a thread definition file manually. The thread definition file is based on XML and has the following structure:

```
<threadsDefinition>

<options>

    <ignoreElement value="SessionID" />
    <ignoreElement ... />
    <ignoreAttribute ... />
    ...
</options>
<threads>
    <thread>
        <test url=control" name="test1" />
        <test ... />
        ...
    </thread>
    <thread>
        <test ... />
        <threads>
            ...
        </threads>
        ...
    </thread>
</threads>
</threadsDefinition>
```

## Options

At the beginning of the threads definition, you can define ignore options. These are definitions for the comparison of test results which elements or attributes should be ignored (see also [Adding Options to a Test Suite](#)).

Within the options element, you can make the following definitions:

Element	Description	Allowed Values / Example
<b>ignoreElement</b>	Ignore the XML element defined in attribute "value" on test result comparison.	If the specified element or attribute does not exist, nothing happens.
<b>ignoreAttribute</b>	Ignore the XML attribute defined in attribute "value" on test result comparison.	

## Tests



Within a thread you can define again threads.

Tests can have the following XML attributes:

Attribute	Description	Mandatory	Allowed Values / Example	
url	Reference to the service you want to call with this test. The service details (URL, port, ...) are defined in the <b>ParallelRequest</b> test.	✔	control	Test calls service 1.
			soap	Test calls service 2.
name	Specify a test name. This name will also be used to name the test error logs and response logs.  If performing a POST request, you need to supply the POST body in an XML file <name>.xml. This file must be stored in folder <b>Request Files</b> as defined in the <b>ParallelRequest</b> test.	✔		
method	HTTP method. Method POST needs an input file in the source folder. <b>The page BRIDGE:@self was not found</b> -- Please check /update the page name used in the MultiExcerpt-Include macro		POST	POST request (default).
			GET	GET request.
path	Supply an appendix to the service path if necessary.		/sessions	
variable	Specify the name of the variable to "store to" or "retrieve from" the variable store. Refer to <a href="#">Working With a Variable Store</a> below for more details.		sessionID	
xpath	Specify an XPath expression to extract a value from the request response to store it to the variable store. Refer to <a href="#">Working With a Variable Store</a> for more details.			

## Working With a Variable Store

You can use a variable store with **ParallelRequest** tests to pass data from one test to others.



In contrast to e.g. [ServiceCommand tests](#), you can only store one variable value to a variable store with ParallelRequest tests.

- One test from the thread definitions stores a value from the test response to the variable store using
  - xpath** to extract the value from the response
  - variable** to specify the name of the variable and the name of the variable store the value is stored to
- A later test replaces placeholders in the request file with values from the variable store using
  - variable** to load the variable value from file

This way, the variable value is used in the request of the second test.

## Storing Variables to a Variable Store

Specify **xpath** and **variable** in a ParallelRequest test. The value extracted by the XPath expression will be stored in the XML file specified by **variable** under name of **variable**. You need at least one ParallelRequest test with this settings to store a value.

## Retrieving Variables From a Variable Store

Specify an existent variable store (<**variable**>.xml) in one or more subsequent ParallelRequest test to retrieve values that have been previously stored. All occurrences of variables in form of **\${variable}** in a given request file will be replaced by the value from the variable store.



The following applies if your variable definitions are not consistent:

- If the variable cannot be found in the variable store, the variable value will be returned as an empty string.
- If you specify **xpath** but no variable store can be found, nothing will happen.



Thus, you can only use variable stores with requests that use a request file. You cannot replace path variables of a REST request with values from a variable store.