IBM Rational Build Forge

Advanced topics: Adaptors

February 11, 2011

Version 1.0

Author:
Kristofer Duer

Note
Before using this information and the product it supports, read the information in “Notices,” on page 19.

Table of Contents

Introduction

An adaptor is much more than just complex execution flow and decision making in IBM Rational Build Forge. It can also support the following features:

- Emailing multiline variables to users
- Dynamic notification groups
- Custom BOM additions
- Project execution control

You can use some or all of these features in a single adaptor.

This document shows how varied and powerful adaptors can be and how adaptors can help accomplish a certain goal within Rational Build Forge. The document does not discuss the details of the adaptor XML; see the online help for those details. Also, this document does not cover every adaptor template currently available with Rational Build Forge. For information about the adaptors, see the descriptions in the adaptors. After you read this document, you should be able to read an adaptor and understand how it works and potential modification points.

Adaptor versus project

A Rational Build Forge project and an adaptor are very similar. For example, a project can do most of what an adaptor can do, and vice versa. There are some significant differences in how adaptors and projects accomplish tasks though. A full understanding of these differences helps determine when to use adaptors in builds instead of project steps.

Entry points

Both an adaptor and a project have an entry point, or first step that runs. For a project, this entry point is typically step 1. For an adaptor, the entry point is defined within the template, or the first run command in the template. However, an adaptor can have multiple entry points. You define these points using the <interface/> block. In an adaptor, these entry points are self-contained execution flows that have the benefit of a shared adaptor definition in the Rational Build Forge environment.

Result evaluation differences

A project has two ways to control success, failure, or warning status: exit codes and log filters. In comparison, an adaptor must be told whether it fails; a pass is assumed unless a fail condition defined in the template is met. This fail condition is specified in the adaptor template through the <step/> element, which is covered later. You base the fail condition on a variable's contents—whether it is empty, has text, or even equality and negation. This simple difference allows for full control of the failure criteria on a variety of factors beyond simple exit codes or pattern matching.

Flow differences

A project flow is somewhat static. The introduction of Job Process Optimization (JPO) certainly allows for a more dynamic flow, however the flow is still largely static. An adaptor conversely can run commands dynamically, but it can also run the same command repeatedly against each line of output.

With steps, flow control options include:

- The IF…ELSE step type, which allows for evaluation of a condition, similar to programming language branching logic.
- The WHILE step type, which offers the ability to continuously cycle through a step command until a condition is no longer true.

An adaptor controls flow using pattern matching against each line of output.

State information differences
A project uses environment variables as the state information for a build. To pass state information between steps, you can use either environments or registers.

An adaptor uses parameters and environment variables to control and send state information to other commands. The parameters are the results of a pattern match, which can allow for incredibly complex, or simple, parameterization of a command result.

ENV differences
Projects and adaptors have similarities when dealing with environment variables. Both can set the main database record, a build environment, or a temporary step\adaptor type variable. The main difference between the two is how they accomplish the goal. A project sets one name=value pair at a time, while an adaptor can dynamically create a name=value pair from a pattern match. Adaptors support multiline variables, whereas projects do not. This feature is helpful when using a large variable of multiple lines in an email notification: you can continuously build the variable during the adaptor run, and at the end have the adaptor use this variable in a notification email.

Bill of Materials (BOM) handling differences
A project affects the BOM using dot commands such as .scan and .bom. These commands usually process only one line of input. An adaptor uses pattern matching to add to the BOM. As a result, each and every line of output from the command could potentially be added to the BOM inside of an adaptor. A project cannot do this. You can use this feature, for example, in continuous-integration adaptors to capture change sets between versions and place them in the BOM.

Notification group differences
A project uses static (predefined) notification groups that you explicitly assign to project completion states such as pass, start, fail, step pass, and so on. An adaptor has the ability to dynamically create a notification group and send out a custom email to that group. This feature does not require the user to exist in Rational Build Forge; it requires only that the SMTP server know the username supplied in the TO: list.

Server usage difference
A project typically uses a selector to determine the server to run on. There are ways, such as .bset buildserver, to specify a particular server to use, but in general selectors choose the servers.

An adaptor, by comparison, does not use a selector. It assigns the server explicitly by name or through a variable.

An adaptor can run commands locally on the management console *without* a local agent.

Adaptor links versus .source, .defect,. test and .pack
Adaptor links join adaptors and projects in a special kind of execution flow. Their sole purpose is to decide whether a build should be rolled back. In other words, adaptor links are the gateways for deciding whether a build should run the following steps. This decision is based on the status of the adaptor (Pass or Fail). If the adaptor passes, then the job is allowed to run. If the adaptor fails, the job is prevented from running and all evidence that the job ran is removed from the system. This is a powerful feature because you specify the criteria for an adaptor passing or failing.

If the adaptor fails, the link purges the current job and rolls back the build tag if this job is the only job for the project that is currently running. If there is another job currently running for the same project, regardless of whether it is an adaptor-linked project, the adaptor link does not roll back the build tag. It still purges the failed job. You can use the Run Limit setting for the project to ensure that never more than one job runs at one time.

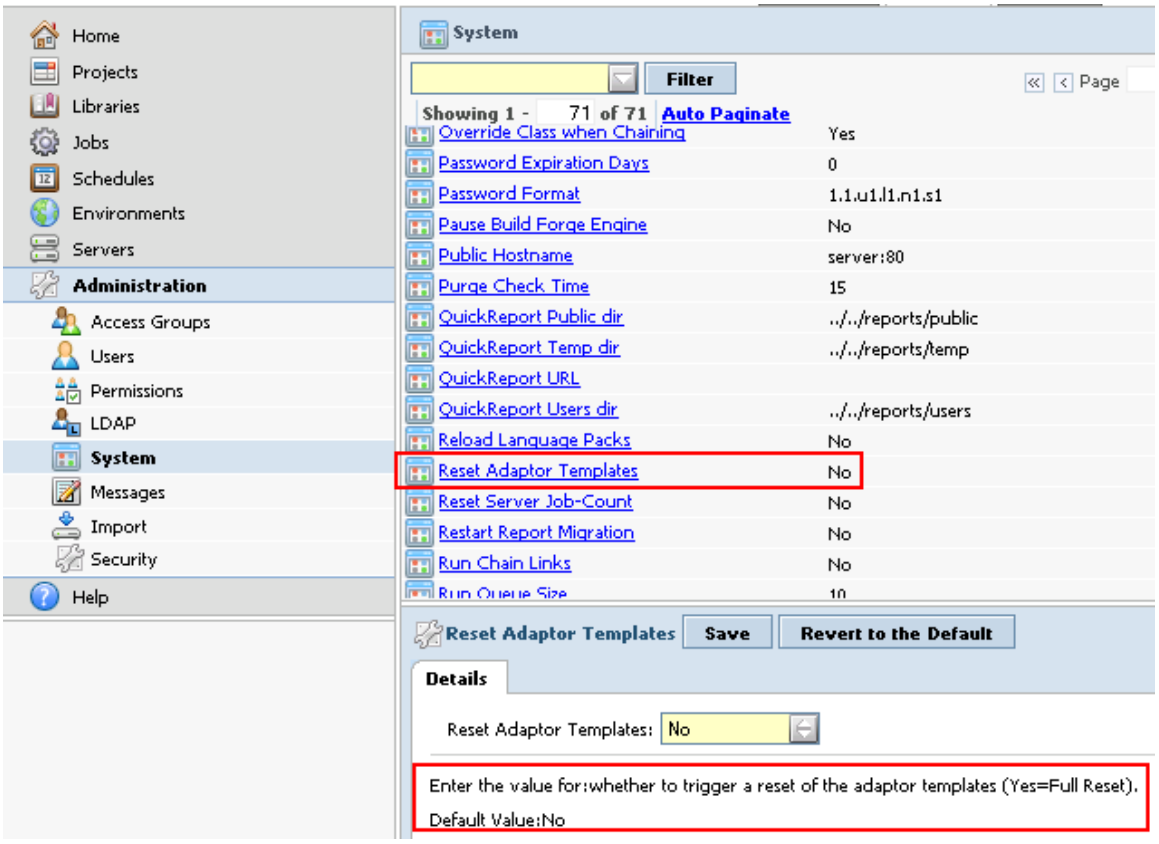
Set the Run Limit setting to 1 for all adaptor link builds to control concurrency issues like the one just described. You can also use semaphores in the first step of a job to ensure that only one instance runs at a time.

You can control whether a job is purged by using the environment variable _CI_BUILD_KEEP. Setting this variable to 1 prevents the build purge on a failed adaptor run. This variable is typically used in debugging scenarios rather than during normal project runs.

Instead of using an adaptor link to call an adaptor, you can use the .source command. However, .source does *not* prevent a build from running. The .source command is truly a step command; if it fails, then the step's halt\continue status determines whether the build progresses. Any steps that occur before the .source step still run. An adaptor link, however, prevents the entire build from running if the adaptor fails. There is no "continue on fail" status for an adaptor link step.

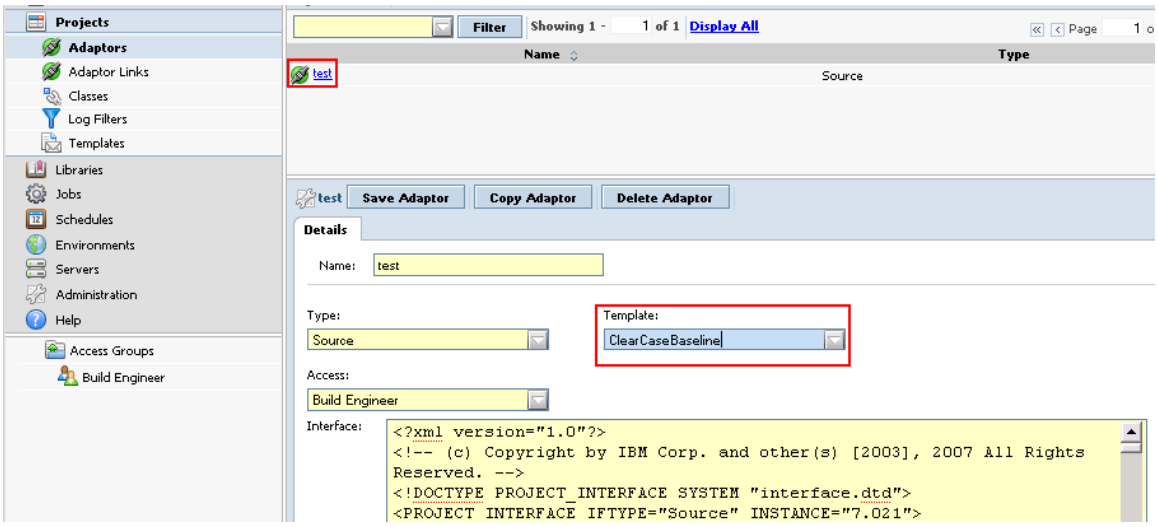
Template vs instantiated

The adaptors provided with the system are designed to be functional examples of how to use adaptors. They are in the interface directory found in %BF_HOME% on computers running Microsoft Windows and in $BF_HOME/Platform on computers running operating systems based on UNIX.. To load edits of these files into Rational Build Forge, use the **Admin > System** option Reset Adaptor Templates, shown here:



Any new adaptors or changes to the default adaptors are assimilated into the system upon a full reload.

These adaptors are just templates until they are either linked to a project or instantiated as a new adaptor in the system. The adaptor link allows for use of either of these artifacts within Build Forge. You instantiate an adaptor by creating a new adaptor from a template or by creating a new one from scratch. Instantiating an adaptor places the new adaptor in an editable location away from the adaptor template and allows for customization of the adaptor text. In the example below, a new adaptor is instantiated using the text from the adaptor template ClearCaseBaseline.



Now any changes that occur for the adaptor instance test will not affect the text contained within the adaptor template ClearCaseBaseline. The distinction between a template and an instantiated adaptor is an important one. The decision of which to use is largely left to the implementation; however it is more common to use instantiated adaptors as opposed to adaptor templates to protect the text within from an upgrade process, or inadvertently reloading the adaptor templates through the option mentioned above.

Basic structure

To describe the basic structure of an adaptor, it is convenient to show a simple example first, and then build on it.

An adaptor uses XML to tell the system how the adaptor should behave. In XML, an element can contain attributes specific to that element, and other child elements with their own attributes, which in turn can contain their own child elements and attributes and so forth. The XML elements are discussed below.

The basic components of an adaptor required for base functionality are:

- `<PROJECT_INTERFACE/>`
  - `<interface/>`
    - `<run/>`
  - `<command/>`

There are more components, but these are the components used by all adaptors to control basic execution flow.

<PROJECT_INTERFACE/>

The <PROJECT_INTERFACE/> element represents the main container for the adaptor. All other adaptor elements are either children or grandchildren of <PROJECT_INTERFACE/>. There are four element types that can exist within a PROJECT_INTERFACE:

one or more <template/> elements
one or more <interface/> elements
one or more <command/> elements
one or more <bomformat/> elements

The <PROJECT_INTERFACE/> element has to attributes you must define: IFTYPE and INSTANCE.

PROJECT_INTERFACE attributes:
The IFTYPE, or interface type, attribute describes what kind of adaptor this is. There are four valid values:

- Source
- Defect
- Test
- Package

The case of the IFTYPE is important.

The INSTANCE attribute refers to the version of Rational Build Forge the adaptor was built against.

<interface/>
The <interface/> element represents the entry point for an adaptor and is as close to a project definition as adaptors get. The element contains the order of steps to run for this particular entry point. In addition, <interface/> contains the decision-making power of pass and fail, and what the adaptor should do in either of those events regarding notifications. A later section discusses how to use multiple <interface/> elements to describe multiple entry points in the same adaptor.

<run/>
The <run/> element has attributes that include the command to be run, what parameters to pass to the command, and optionally a condition to determine whether the command runs.

<command/>
The <command/> element represents a "step" within the adaptor. The command can be called multiple times from different <interface/> elements, or just once as a simple step would be in a project run. The power of the adaptor lies within the <match/> child element within the <command/> element, which will be covered later.

The sample adaptor so far looks like this now:

```
<PROJECT_INTERFACE IFTYPE="Source" INSTANCE="7.02">
<!-- Adaptor sample. -->

<interface>
      <run command="echo" params="" server="$BF_SERVER" dir="" timeout="360"/>
</interface>

<command name="echo">
      <execute>echo hello world</execute>
</command>
</PROJECT_INTERFACE>
```

This adaptor will always pass; however, it is still useful to illustrate that an adaptor only requires these bare components. Everything else is optional.

Step log excerpt:

```
INT     Creating Adaptor Decision Logic.
INT     Preparing Adaptor command set.
INT     Using default adaptor entry point.
INT     Preparsing Command Set: [echo hello world], using Params: ''''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder854671951bomp2]
INT     Command Set Parsed To: [echo hello world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder854671951bomp2]
EXEC    STARTBFBomPlaceholder854671951bomp2
EXEC    hello world
EXEC    ENDBFBomPlaceholder854671951bomp2
```

Flow control with adaptors

Flow control within an adaptor is controlled through the <resultsblock/> children and grandchildren.

- <PROJECT_INTERFACE/>
  - <interface/>
    - <run/>
  - <command/>
    - **<resultsblock>**
      - **<match/>**

<resultsblock/>

The <resultsblock/> element acts as a parent container for what to do with the command response. Typically, you will have one or more <match/> blocks within a single <resultsblock/> to help determine what the next action is, based on the result. Each line of output from the command is evaluated by each of the <resultsblock/> children.

<match/>

The <match/> block is arguably where the adaptors power lies. This simple block allows the adaptor to run a regular expression against each line of output and perform a task based on the match. A task can be to set an environment variable, run another command, or set some information in the BOM. In addition, portions of the match can be sent to the task to be performed. The semantics follow the Perl regular expression paradigm. For example, (hello)\s(world) matches the word "hello" followed by a space and then by the word "world." In addition, because of the parentheses (), the internal variable $1 becomes hello, and $2 becomes world. Note that the parameters exist only as $1, $2, and so on in the current command. After they are passed as parameters to the next action, the order in which they are passed determines what will be $1, $2, and so on in the next action. For example, consider using that match with this action:

```
<run command="echo2" params="$2,$1" server="$BF_SERVER" dir="" timeout="360"/>
```

The action places hello into $2 and world into $1 for the next command, even though in the current command hello is $1 and world is $2 based on the match pattern.

The sample adaptor so far looks like this:

```
<PROJECT_INTERFACE IFTYPE="Source" INSTANCE="7.02">
<!-- Adaptor sample. -->

<interface>
      <run command="echo" params="" server="$BF_SERVER" dir="" timeout="360"/>
</interface>

<command name="echo">
      <execute>echo hello world</execute>
            <resultsblock>
                  <match pattern="^(hello)\s(world)$">
                        <run command="echo2" params="$1" server="$BF_SERVER" dir=""
timeout="360"/>
                  </match>
            </resultsblock>
</command>
<command name="echo2">
      <execute>echo $1</execute>
</command>
</PROJECT_INTERFACE>
```

To better illustrate the results, an additional command is used. It receives a parameter and uses the parameter in running a command. The adaptor is now using the results by running a second command based on the match. Much like the first example, this adaptor will always pass.

Step log excerpt:

```
INT     Creating Adaptor Decision Logic.
INT     Preparing Adaptor command set.
INT     Using default adaptor entry point.
INT     Preparsing Command Set: [echo hello world], using Params: ''''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder1288390126bomp2]
INT     Command Set Parsed To: [echo hello world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder1288390126bomp2]
EXEC    STARTBFBomPlaceholder1288390126bomp2
EXEC    hello world
EXEC    ENDBFBomPlaceholder1288390126bomp2
RESULT  0
INT     Line [hello world] matched pattern '^(hello)\s(world)$'.
INT     Preparsing Command Set: [echo $1], using Params: ''hello''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder207947636bomp4]
INT     Command Set Parsed To: [echo hello]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder207947636bomp4]
EXEC    STARTBFBomPlaceholder207947636bomp4
EXEC    hello
EXEC    ENDBFBomPlaceholder207947636bomp4
RESULT  0
```

Declaring fail criteria

Determining the criteria for a failure is central to designing an adaptor and is based solely on the use case the adaptor is built against. Failure criteria can be items such as were there any change sets, is there a build record, has file size changed, and so on. By default, an adaptor passes. Consequently, you must know what will constitute a failure for a particular use case.

There two main components to determine a failure are <setenv/> and <ontempenv/>.

- <PROJECT_INTERFACE/>
  - <interface/>
    - **<ontempenv/>**
    - <run/>
  - <command/>
    - <resultsblock>
      - <match/>
  - **<setenv/>**

<ontempenv/>

The <ontempenv/> element allows the adaptor to set pass or fail status based on evaluating a condition on one of four possibilities:
TRUE comparison
FALSE comparison
hastext comparison
isempty comparison

These four options give an adaptor decision making ability that is much more flexible than simple exit codes and filter matching. The most common use for the evaluation is an environment variable's contents, or lack of

contents. For example, the “isempty” and “hastext” comparisons can determine pass or fail based on whether a variable has any text within it. A common use case is to fill a temporary variable during an adaptor run only if source changes are detected. This variable is then used in a “isempty” comparison; if it is empty, then no source changes occurred and the adaptor fails.

This is the first place the condition attribute is introduced. The condition attribute can be used in a number of places including <setenv/>, <run/>, <bom/>, <adduser/> and <requeue/>. Some of these elements have not been discussed yet; however, it is good to note that the condition attribute can be used in these locations. For more information about the condition attribute, see the online help or this tech note: http://www-01.ibm.com/support/docview.wss?uid=swg21303318.

<setenv/>

With the <setenv/> element, you can control environment variables within the adaptor. There are three types of environment variables you can use within an adaptor: adaptor scope, build scope, and master environment record scope. The same variable name can be used in each of these scopes within the same adaptor. The online help describes how to modify the use of <setenv/> to influence each type of variable. The keys to note are:

- Adaptor-level variable: Exists only for the adaptor itself; it is not passed to the build
- Build level: Exists only for the steps following the adaptor; it is not available to the adaptor
- Master environment record: Only exists for builds following this one; it is not available to the current adaptor or current build

It is easy to see why the same variable would need to be set using multiple <setenv/> blocks to provide availability at different levels of the overall build flow.

The sample adaptor has gained a few more attributes and now is given fail criteria. In this example, the fail criterion is based on the existence of “hello world” in the output of the command. Because the command echoes “hello world” this adaptor always fails. The point however is that, with essentially the same adaptor, adding in a fail condition can modify the outcome dramatically.

```
<PROJECT_INTERFACE IFTYPE="Source" INSTANCE="7.02">
<!-- Adaptor sample. -->

<interface>
      <run command="echo" params="" server="$BF_SERVER" dir="" timeout="360"/>

      <ontempenv name="FOUND" state="hastext">
            <step result="FAIL" />
      </ontempenv>
</interface>

<command name="echo">
      <execute>echo hello world</execute>
            <resultsblock>
                  <match pattern="^(hello)\s(world)$">
                        <run command="echo2" params="$1" server="$BF_SERVER" dir=""
timeout="360"/>
                  </match>
            </resultsblock>
</command>
<command name="echo2">
      <execute>echo $1</execute>
            <resultsblock>
                  <match pattern="^(hello)">
                        <setenv name="FOUND" value="TRUE" type="temp" />
                  </match>
            </resultsblock>
```

```
</command>
</PROJECT_INTERFACE>

INT     Creating Adaptor Decision Logic.
INT     Preparing Adaptor command set.
INT     Using default adaptor entry point.
INT     Preparsing Command Set: [echo hello world], using Params: ''''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder370549299bomp2]
INT     Command Set Parsed To: [echo hello world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder370549299bomp2]
EXEC    STARTBFBomPlaceholder370549299bomp2
EXEC    hello world
EXEC    ENDBFBomPlaceholder370549299bomp2
RESULT  0
INT     Line [hello world] matched pattern '^(hello)\s(world)$'.
INT     Preparsing Command Set: [echo $1], using Params: ''hello''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder120973912bomp4]
INT     Command Set Parsed To: [echo hello]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder120973912bomp4]
EXEC    STARTBFBomPlaceholder120973912bomp4
EXEC    hello
EXEC    ENDBFBomPlaceholder120973912bomp4
RESULT  0
INT     Line [hello] matched pattern '^hello'.
INT     Temp variable 'FOUND' set to [TRUE].
INT     Text Action on var [FOUND] set step state to 'F'.
```

BOM management

An important feature subset of adaptors is their ability to populate the build's bill of materials with important snapshot information. In some cases, this involves change sets; in others, it is defect records and status. However, in all cases, the BOM information is a collection of vital information to the build. The actual information captured largely depends on the use case. This section covers how the mechanics of the adaptor BOM elements work. There are quite a few elements used to make a BOM entry from an adaptor: <bom/>, <field/>, <bomformat/>, and <section/> as depicted below.

- <PROJECT_INTERFACE/>
  - <interface/>
    - <ontempenv/>
    - <run/>
  - <command/>
    - <resultsblock>
      - <match/>
      - **<bom/>**
        - **<field/>**
  - <setenv/>
  - **<bomformat/>**
    - **<section/>**
      - **<field/>**

<bom/>

The <bom/> element places a piece of text, or more often a regex variable return, into a bom field. Because they are found in a <match/> element, the regex captured can be used to run another command in addition to populating the bom. This behavior introduces the concept of multiple child elements within a single <match/>

element, each receiving the same results. There are three attributes the <bom/> element uses: category, section, and an optional condition.

<bomformat/>
The <bomformat/> element is the parent container for descriptors that determine how a BOM is stored and displayed. In most cases, this element is the top level expandable item found in the user interface. However, as we will see later, even <section/> attributes can be expanded. There are two attributes found within each <bomformat/> element: category and title. The category is the internal mechanism used by the <bom/> element to match the child fields; while title is the title that is displayed in the user interface. There are one or more <section/> child elements within each <bomformat/> element that further break down how the BOM is displayed.

<section/>
The <section/> element is the parent container for the <field/> elements that describe the columns and order of display. The <section/> is mainly meant to be a logical grouping of information and is where any nested relationships are defined. Three attributes are available for each <section/>: name, and optionally expandable and parent. The ClearQuestClearCaseByActivity adaptor has an example of using the last two optional attributes to make nested sections.

<field/>
The meaning of the <field/> element depends on its parent element. With a <bom/> element as the parent, the <field/> element maps to a <field/> element described in the <bomformat/> section. In this event, you must define two attributes:

- name: The name of the target <field/> element
- text: The value to place in the BOM field

With a <section/> element as the parent, the <field/> element describes a column for the BOM. In this event, you must define three attributes:

- order: The order the field will appear in within the user interface
- name: The internal name mechanism a <field/> element found under <bom/> uses to identify this field
- title: The display text used in the user interface

The sample adaptor is getting more complicated. First, we changed the fail condition to be isempty because we needed a passing adaptor at this point. Then, we added two <bomformat/> elements to show how to fill using pattern matching and to illustrate how $1 gets overwritten within a <resultsblock/>. Here we have the first command the same as it always was: echo hello world. The first BOM entry captures both of these words and places hello into field 1 of category SAMPLE section echo, and world into field 2 of the same. The next run command pulls out each word individually and adds them to the single field in SAMPLE2 category echo. This is an overly simple example as it is intended for teaching purposes; see the current adaptor templates to get more in-depth examples of multiple BOM entries from different patterns from the same command, or even nested sections.

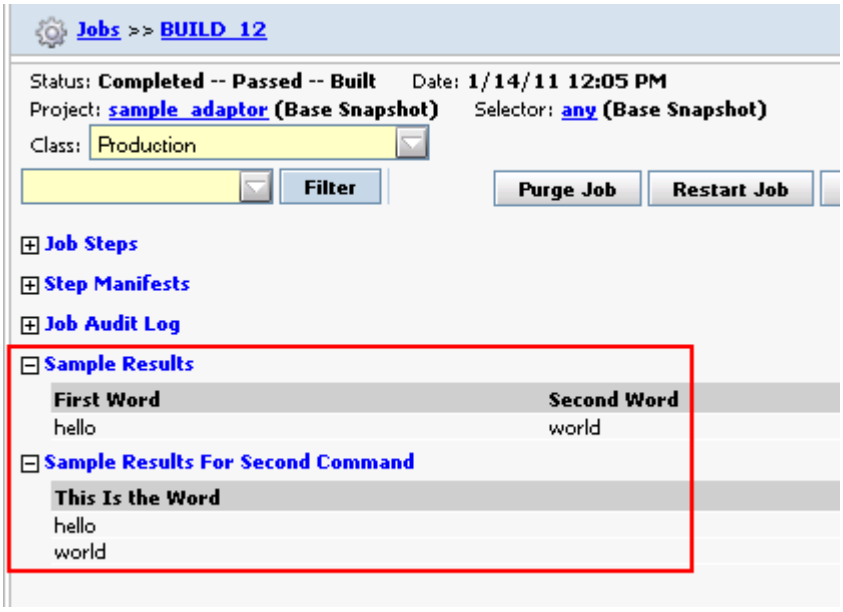
```
<PROJECT_INTERFACE IFTYPE="Source" INSTANCE="7.02">
<!-- Adaptor sample. -->
<interface>
      <run command="echo" params="" server="$BF_SERVER" dir="" timeout="360"/>

      <ontempenv name="FOUND" state="isempty">
            <step result="FAIL" />
      </ontempenv>
</interface>
<command name="echo">
      <execute>echo hello world</execute>
```

```
            <resultsblock>
                  <match pattern="^(hello)\s(world)$">
                        <run command="echo2" params="$1" server="$BF_SERVER" dir=""
timeout="360"/>
                        <run command="echo2" params="$2" server="$BF_SERVER" dir=""
timeout="360"/>
                        <bom category="SAMPLE" section="echo">
                              <field name="first" text="$1"/>
                              <field name="second" text="$2"/>
                        </bom>
                  </match>
            </resultsblock>
</command>
<command name="echo2">
      <execute>echo $1</execute>
            <resultsblock>
                  <match pattern="^(.+?)$">
                        <setenv name="FOUND" value="$1" type="temp" />
                        <bom category="SAMPLE2" section="echo">
                              <field name="word" text="$1"/>
                        </bom>
                  </match>
            </resultsblock>
</command>
<bomformat category="SAMPLE" title="Sample Results">
      <section name="echo">
            <field order="1" name="first" title="First Word"/>
            <field order="2" name="second" title="Second Word"/>
      </section>
</bomformat>
<bomformat category="SAMPLE2" title="Sample Results For Second Command">
      <section name="echo">
            <field order="1" name="word" title="This Is the Word"/>
      </section>
</bomformat>
</PROJECT_INTERFACE>
```

The step log shows the BOM getting populated. The screen shot after the step log shows the two categories and how they display. The field order influences which column is displayed first.

```
INT     Creating Adaptor Decision Logic.
INT     Preparing Adaptor command set.
INT     Using default adaptor entry point.
INT     Preparsing Command Set: [echo hello world], using Params: ''''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder613249343bomp2]
INT     Command Set Parsed To: [echo hello world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder613249343bomp2]
EXEC    STARTBFBomPlaceholder613249343bomp2
EXEC    hello world
EXEC    ENDBFBomPlaceholder613249343bomp2
RESULT  0
INT     Line [hello world] matched pattern '^(hello)\s(world)$'.
INT     Adding 'hello' to BOM category 'SAMPLE' section 'echo' field 'first'.
INT     Adding 'world' to BOM category 'SAMPLE' section 'echo' field 'second'.
INT     Preparsing Command Set: [echo $1], using Params: ''hello''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder1175042181bomp4]
INT     Command Set Parsed To: [echo hello]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder1175042181bomp4]
EXEC    STARTBFBomPlaceholder1175042181bomp4
EXEC    hello
```

```
EXEC    ENDBFBomPlaceholder1175042181bomp4
RESULT  0
INT     Line [hello] matched pattern '^(.+?)$'.
INT     Temp variable 'FOUND' set to [hello].
INT     Adding 'hello' to BOM category 'SAMPLE2' section 'echo' field 'word'.
INT     Preparsing Command Set: [echo $1], using Params: ''world''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder1091771168bomp7]
INT     Command Set Parsed To: [echo world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder1091771168bomp7]
EXEC    STARTBFBomPlaceholder1091771168bomp7
EXEC    world
EXEC    ENDBFBomPlaceholder1091771168bomp7
RESULT  0
INT     Line [world] matched pattern '^(.+?)$'.
INT     Temp variable 'FOUND' set to [world].
INT     Adding 'world' to BOM category 'SAMPLE2' section 'echo' field 'word'.
```



Adaptor template

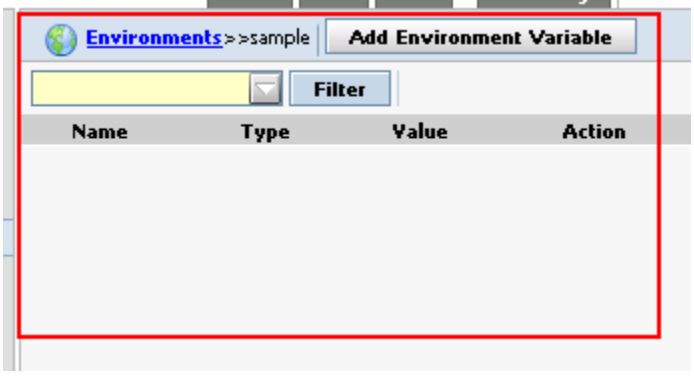
The <template/> section provides the environment variables the adaptor uses. The environment variables are represented by the child element <env/>. Name the variables and assign them values that describe what the variables are used for. The variables are only read and used when an adaptor link is made, linked to an environment, and you click the **Populate Environment** button.

The sample adaptor now has a new <template/> section that populates two variables: VAR1=hello and VAR2=world. In addition, the adaptor has been changed to echo ${VAR1} ${VAR2}, which essentially mimics what we have already seen with this example. The result is the same as before: echo hello world.

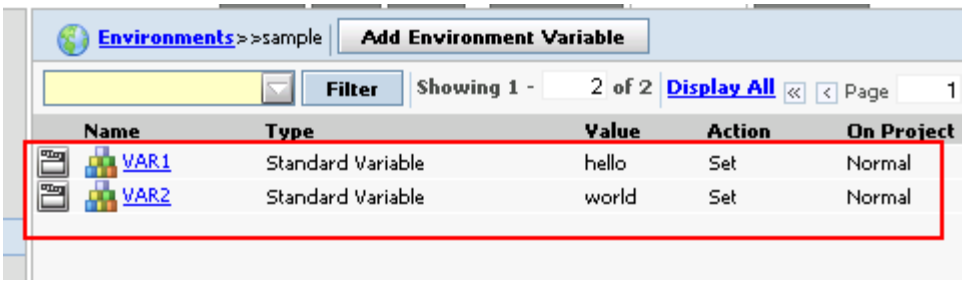
```
<PROJECT_INTERFACE IFTYPE="Source" INSTANCE="7.02">
<!-- Adaptor sample. -->
<template>
      <env name="VAR1" value="hello"/>
      <env name="VAR2" value="world" />
</template>
<interface>
      <run command="echo" params="" server="$BF_SERVER" dir="" timeout="360"/>

      <ontempenv name="FOUND" state="isempty">
```

```
            <step result="FAIL" />
      </ontempenv>
</interface>

<command name="echo">
      <execute>echo ${VAR1} ${VAR2}</execute>
            <resultsblock>
                  <match pattern="^(hello)\s(world)$">
                        <run command="echo2" params="$1" server="$BF_SERVER" dir=""
timeout="360"/>
                        <run command="echo2" params="$2" server="$BF_SERVER" dir=""
timeout="360"/>
                        <bom category="SAMPLE" section="echo">
                              <field name="first" text="$1"/>
                              <field name="second" text="$2"/>
                        </bom>
                  </match>
            </resultsblock>
</command>
<command name="echo2">
      <execute>echo $1</execute>
            <resultsblock>
                  <match pattern="^(.+?)$">
                        <setenv name="FOUND" value="$1" type="temp" />
                        <bom category="SAMPLE2" section="echo">
                              <field name="word" text="$1"/>
                        </bom>
                  </match>
            </resultsblock>
</command>
<bomformat category="SAMPLE" title="Sample Results">
      <section name="echo">
            <field order="1" name="first" title="First Word"/>
            <field order="2" name="second" title="Second Word"/>
      </section>
</bomformat>
<bomformat category="SAMPLE2" title="Sample Results For Second Command">
      <section name="echo">
            <field order="1" name="word" title="This Is the Word"/>
      </section>
</bomformat>

</PROJECT_INTERFACE>
```

Initially, there is a blank environment called sample.



After the adaptor link is made, linked to this environment, and you click the **Populate Environment** button, the sample environment has the adaptors environment with the default values specified in the <template/> section.



The sample step log shows the new variables being used. It also shows essentially the same result occurring:

```
INT     Creating Adaptor Decision Logic.
INT     Preparing Adaptor command set.
INT     Using default adaptor entry point.
INT     Preparsing Command Set: [echo ${VAR1} ${VAR2}], using Params: ''''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder416871492bomp2]
INT     Command Set Parsed To: [echo hello world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder416871492bomp2]
EXEC    STARTBFBomPlaceholder416871492bomp2
EXEC    hello world
EXEC    ENDBFBomPlaceholder416871492bomp2
RESULT  0
INT     Line [hello world] matched pattern '^(hello)\s(world)$'.
INT     Adding 'hello' to BOM category 'SAMPLE' section 'echo' field 'first'.
INT     Adding 'world' to BOM category 'SAMPLE' section 'echo' field 'second'.
INT     Preparsing Command Set: [echo $1], using Params: ''hello''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder883895613bomp4]
INT     Command Set Parsed To: [echo hello]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder883895613bomp4]
EXEC    STARTBFBomPlaceholder883895613bomp4
EXEC    hello
EXEC    ENDBFBomPlaceholder883895613bomp4
RESULT  0
INT     Line [hello] matched pattern '^(.+?)$'.
INT     Temp variable 'FOUND' set to [hello].
INT     Adding 'hello' to BOM category 'SAMPLE2' section 'echo' field 'word'.
INT     Preparsing Command Set: [echo $1], using Params: ''world''.
INT     Command Set Parsed To: [echo STARTBFBomPlaceholder482239853bomp7]
INT     Command Set Parsed To: [echo world]
INT     Command Set Parsed To: [echo ENDBFBomPlaceholder482239853bomp7]
EXEC    STARTBFBomPlaceholder482239853bomp7
EXEC    world
EXEC    ENDBFBomPlaceholder482239853bomp7
RESULT  0
INT     Line [world] matched pattern '^(.+?)$'.
INT     Temp variable 'FOUND' set to [world].
INT     Adding 'world' to BOM category 'SAMPLE2' section 'echo' field 'word'.
```

Additional elements

There are a number of additional elements that influence behaviors, where the adaptor runs, and even custom notification groups. This section highlights these elements. For more information about their usage, see the online help.

<integrate/>: This element is a replacement element for <execute/> and specifically tells Rational Build Forge to run on the local Rational Build Forge engine computer with the path context set in the %BF_HOME %\integration | $BF_HOME/Platform/integration folder depending on which operating system that Rational Build Forge is installed on. This element is used most with the ClearQuest adaptors, as they use the bfcqresolve.pl script found in this directory to operate.

<onproject/>: The <onproject/> element is a parent element used for notification purposes. Use this element to send an email to a custom list of users as well as a custom message based on this past run of the adaptor. There are pass and fail options. The fail option is only used if the adaptor is called with a dot command instead of an adaptor link.

<notify/>: The <notify/> element describes a custom notification template and specifies the group to send the notification to. You can use this element with adaptor temp variables to send a list of change sets or other build-specific information.

<adduser/>: The <adduser/> element dynamically adds users to an access group. This element works only with existing access groups; it does not create access groups.

<relate/>: The <relate/> element associates a user with an artifact that the adaptor captures. The most common use is to relate a user with a particular file they changed. Ideally, a later step in the same project would look for something like “Error file.c” with a log filter set to notify changers. If that is the case, the user associated with the file through the adaptor receives the “notify changers” email from the build. Note that the adaptor only builds the list; it does not act on it. A step in the build must have a filter with the action Notify Changers, as well as a pattern match for the related users to be emailed. Also note that the relationship and the notify changers filter do not have to be the same. If a line matches the filter, then the entire line is scanned for the text captured by the <relate/> tag. For example, if user bob is associated with artifact file.c and a later step matches against “Error” finding “Error compiling file.c,” Rational Build Forge evaluates the relationships. Because the text file.c appears in the line, the filter match occurs and an email is sent to bob.

Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.