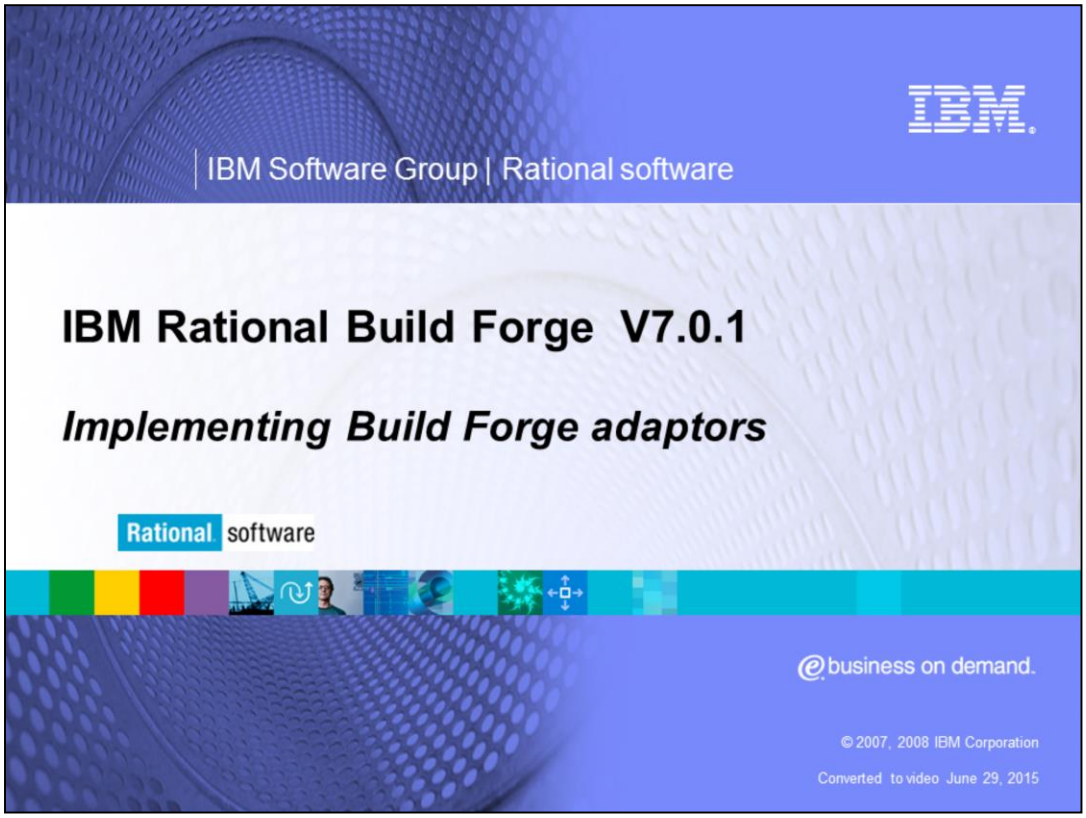
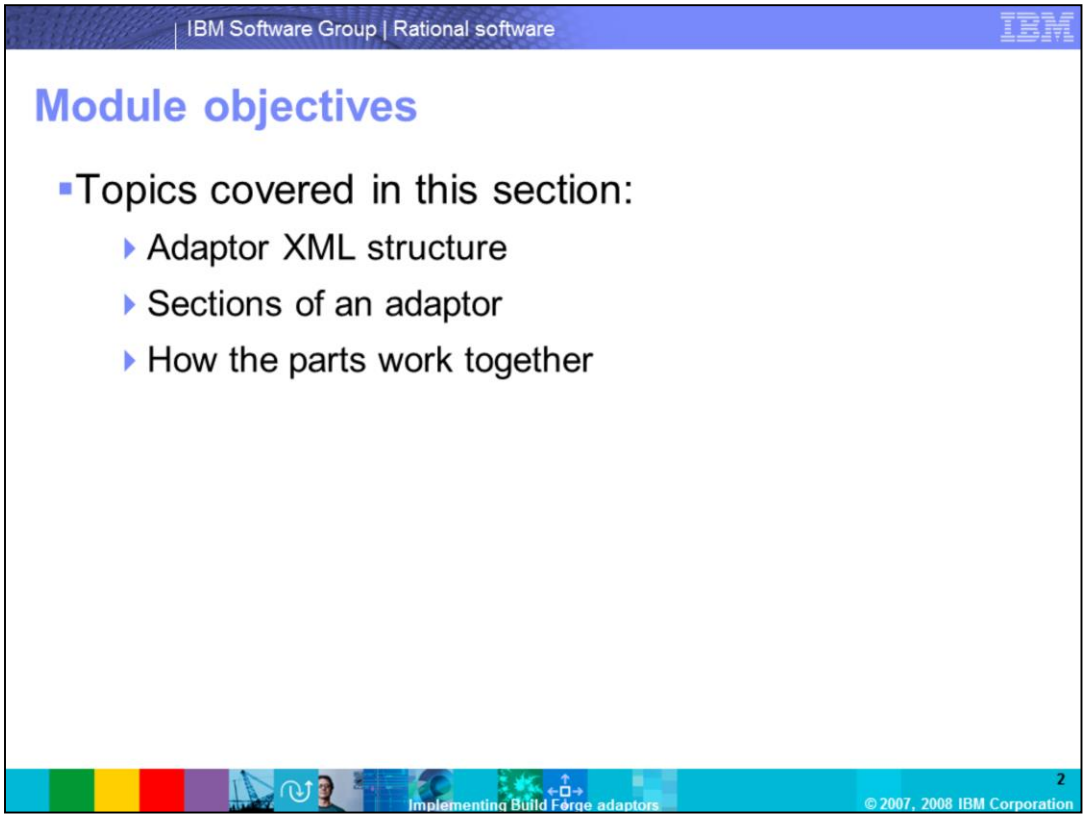


This module will cover implementing adaptors in Build Forge version 7.0.1 and later.

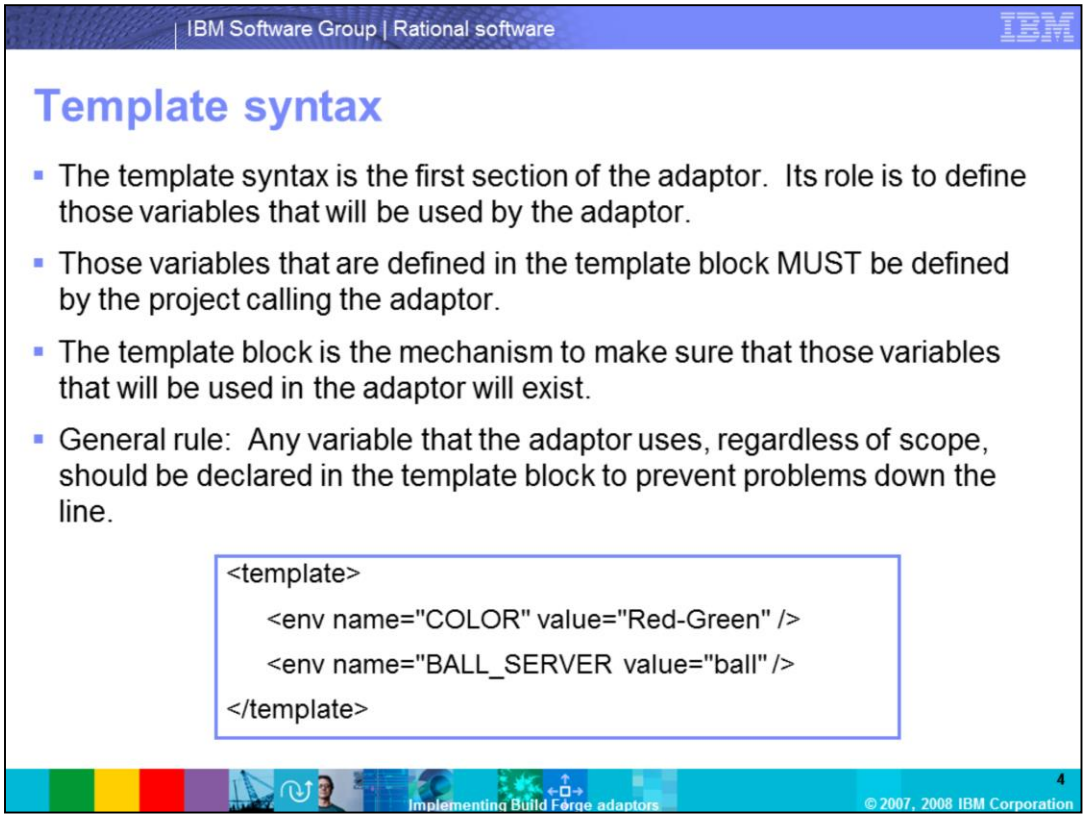


This presentation will cover the Build Forge Adaptor implementation. This module will cover the general structure of an adaptor and how it is set up. From that, details will be provided about each section of an adaptor. Last, the module will provide real examples of how they are implemented. Once you are done with this module you should be able to implement your own adaptor and trace the execution path of an existing adaptor.

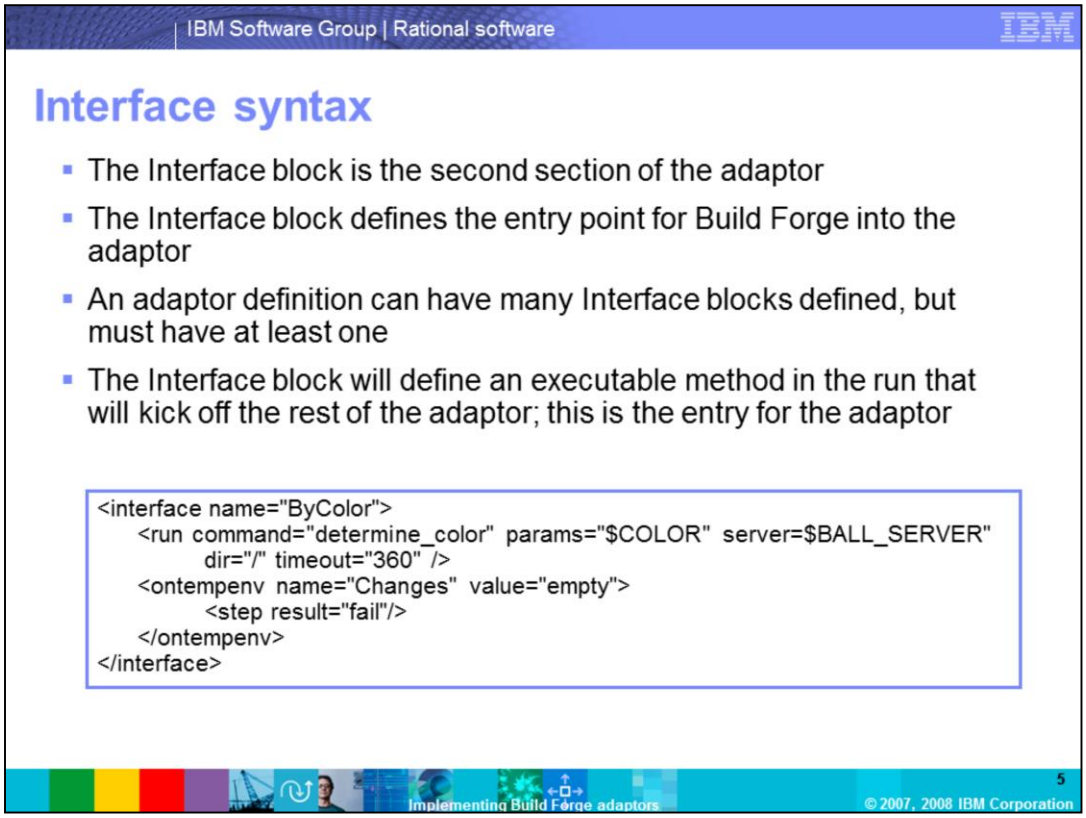
An example adaptor

```
<template>
    <env name="COLOR" value="Red:Green"/>
    <env name="BALL_SERVER" value="ball"/>
</template>
<interface name="ByColor">
<setenv name="Changes" value="" type="temp"/>
    <run command="determine_color" params="$COLOR" server="$BALL_SERVER" dir="/" timeout="360" />
    <ontempenv name="Changes" value="empty">
        <step result="fail"/>
    </ontempenv>
</interface>
<command name="determine_color">
    <execute>
        echo $1
    </execute>
    <resultsblock>
        <match pattern="^(.*?):(.*?)$">
            <bom category="ball_category" section="ball_color">
                <field name="color1" text="$1"/>
            </bom>
            <setenv group="Temp" name="COLOR" value="$2:$1"/>
            <setenv name="Changes" value="TRUE" type="temp append"/>
        </match>
    </resultsblock>
</command>
<bomformat category="ball_category" title="Ball Category">
    <section name="ball_color">
        <field order="1" name="color1" title="Color 1"/>
    </section>
</bomformat>
```

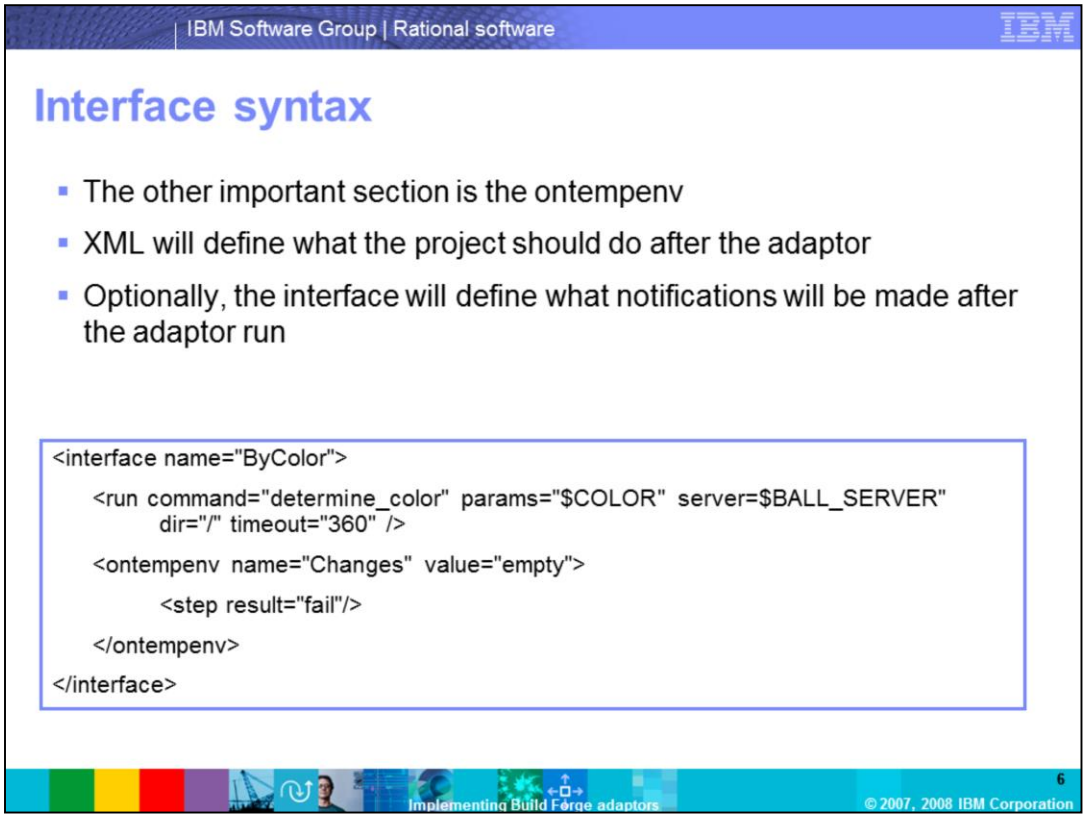

This is the example adaptor from which this presentation will be pulling examples. Do not worry about absorbing it all from this slide as the sections are discussed in the following slides. Note that this is a fully functional adaptor that will work in Build Forge.



The template block is an optional section of the adaptor, but an important one, especially for adaptors that may be deployed by other people. The template block is always declared first if the adaptor has one. The intention of the template is to declare all variables that the adaptor may use or require in addition to defining the default values for those variables. The adaptor does not enforce this template but if another person is going to use the adaptor this would tell that person what environment variables should be in the environment to use this adaptor. Also, when setting up an adaptor link there is a check box to “Populate Env” when creating the link. The populate env option will draw on the variables and default values defined in the template to populate the given environment. To prevent problems later, the best practice suggestion for this section is to declare any variable that the adaptor uses.

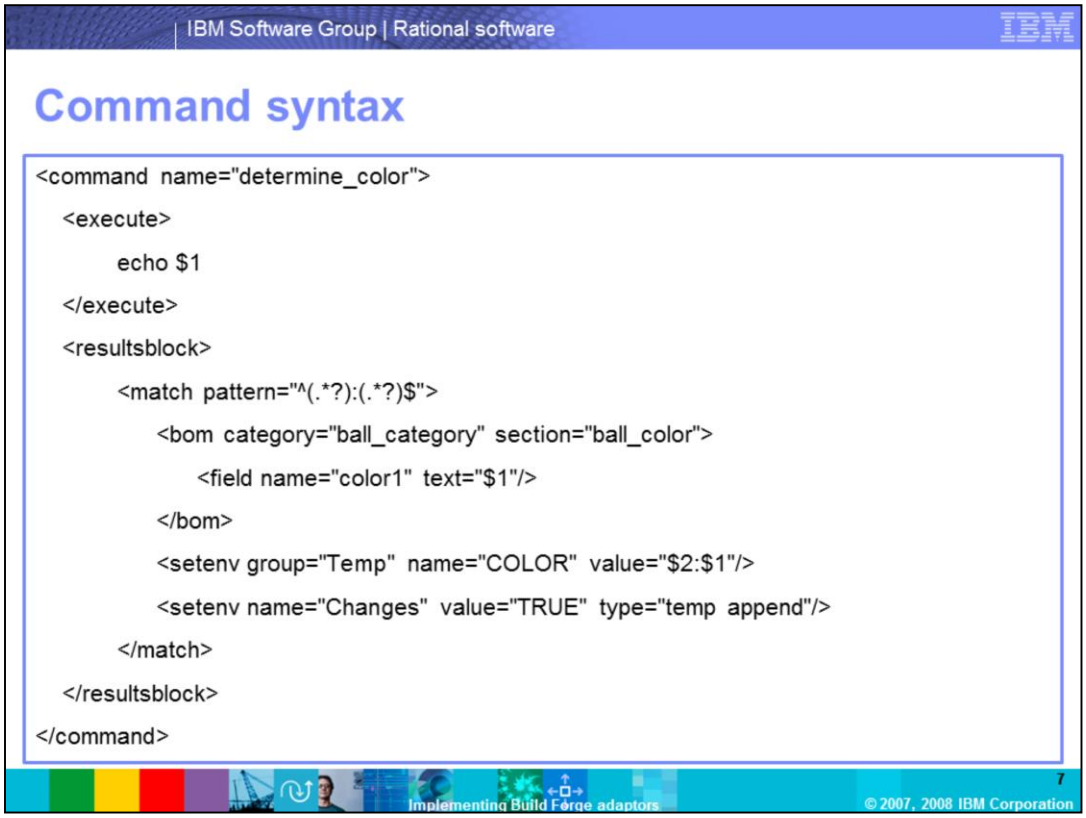


This section is the interface block. The interface block comes right after the template block (if there is a template block defined). The interface block defines where the entry point will be for the adaptor. That is, when the adaptor starts execution it is starting here. Generally, the interface block handles any initialization that needs to happen and defines what commands should start the adaptor. The other important job that the interface has is to determine the pass and fail criteria for the adaptor as a whole. So when you finish the adaptor execution it will return to the interface block to determine if it was a success or a failure.

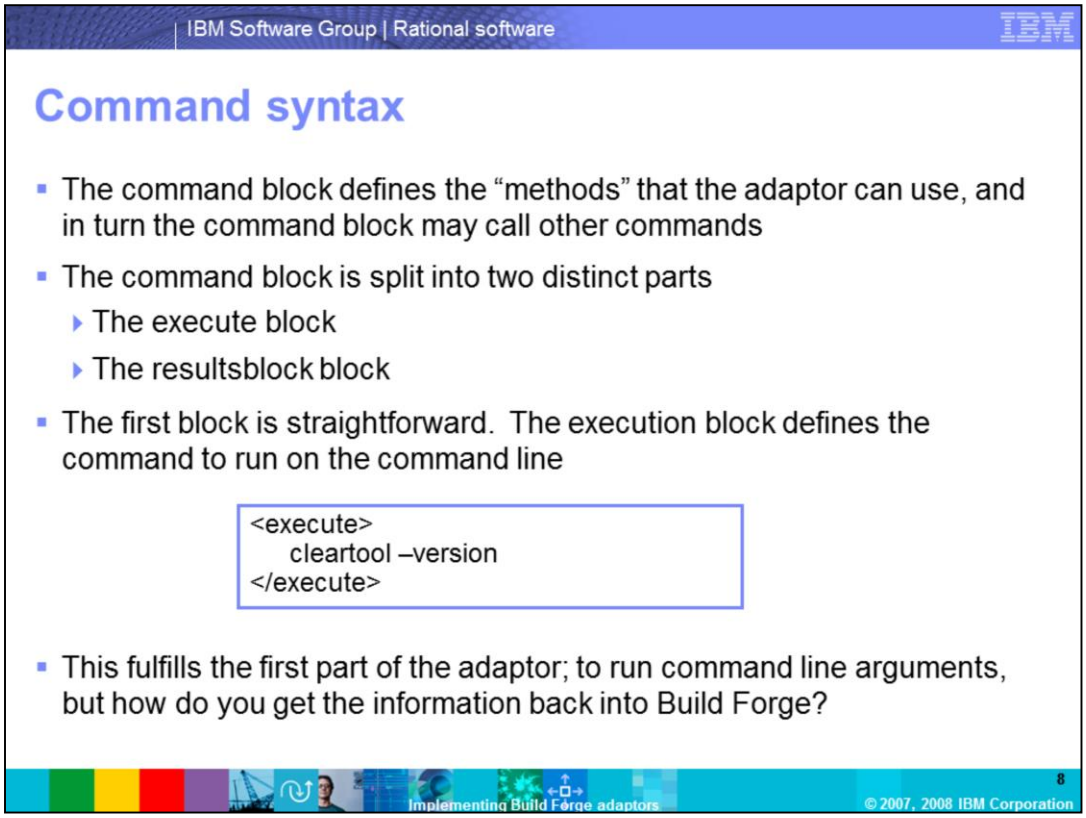


The other important section is the ontempenv. XML will define what the project should do after the adaptor. So in the case where you have defined an adaptor link, the ontempenv will form the if-then-else structure that will decide what action the adaptor will do. Optionally, the interface will define what notifications will be made after the adaptor runs.

Note: There was an architectural change between versions 7.0 and 7.0.1 that affected the interface structure. Before, in 7.0, it was possible to define multiple interface blocks, but in 7.0.1 this was eliminated to only allow an adaptor a single interface block. This change was made to encourage cohesion when defining an adaptor. In 7.0.1 and later versions, the adaptor should have a very granular, specific, role.

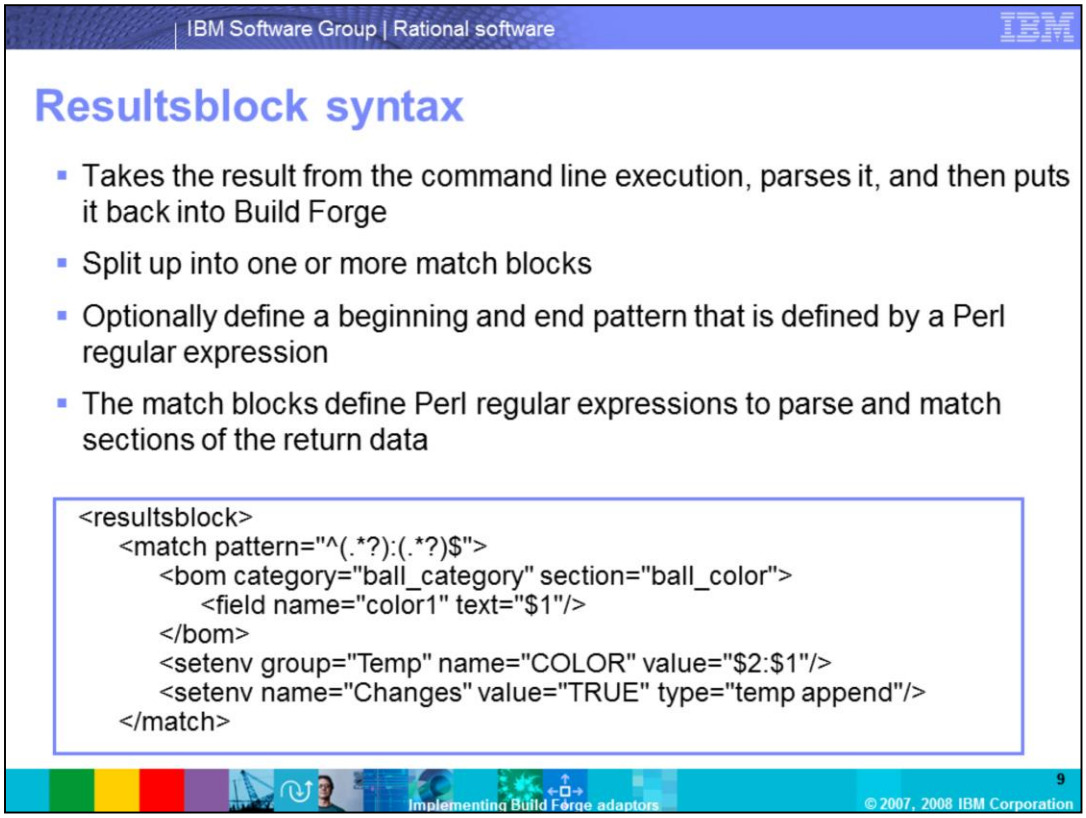


This is the command block. It is the main part of the adaptor.

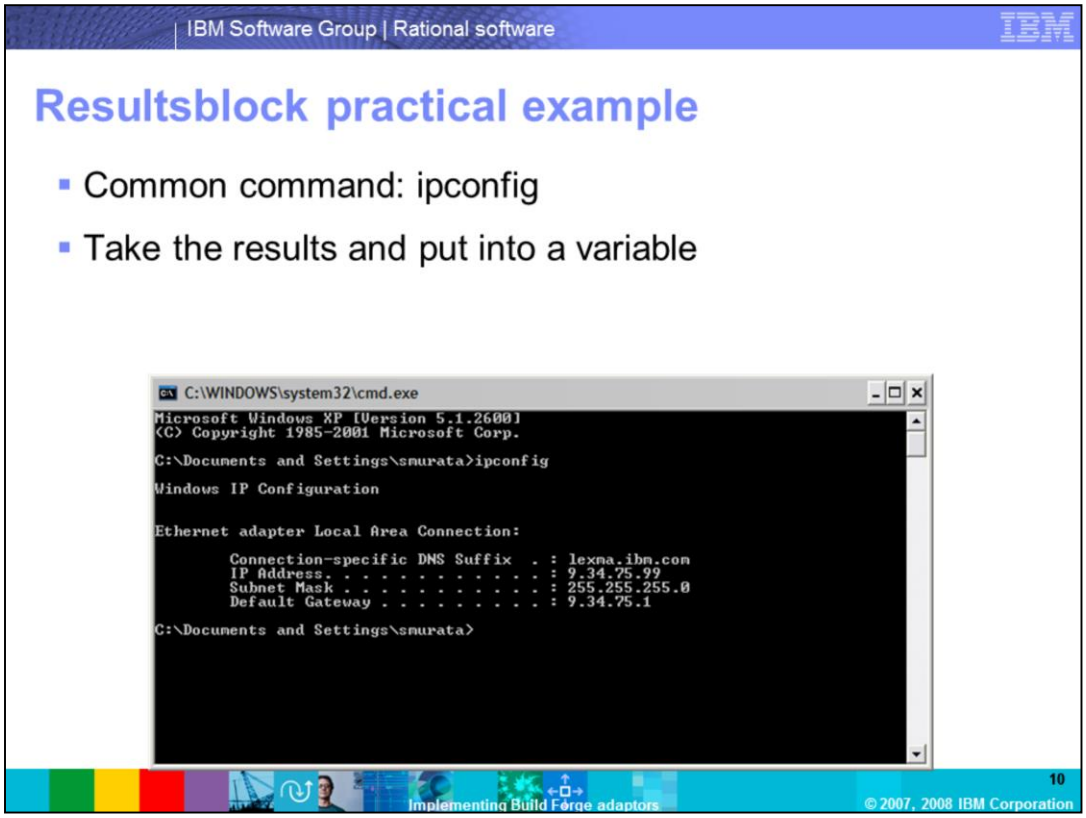


The command block defines the method calls in the Adaptor. The commands are reusable methods that can be used by the Adaptor. The command block defines a name that the Adaptor can use at another point to call this command. Other than that, the command block consists of two other parts: the Execute block and the Resultsblock Block. The execute block is straightforward; it acts the same way that the Build Forge step does. It takes any shell command that can run on the agent, and then runs it. Once the command is run and the results of that command are sent back, then Build Forge moves on to the resultsblock to determine what to do next.

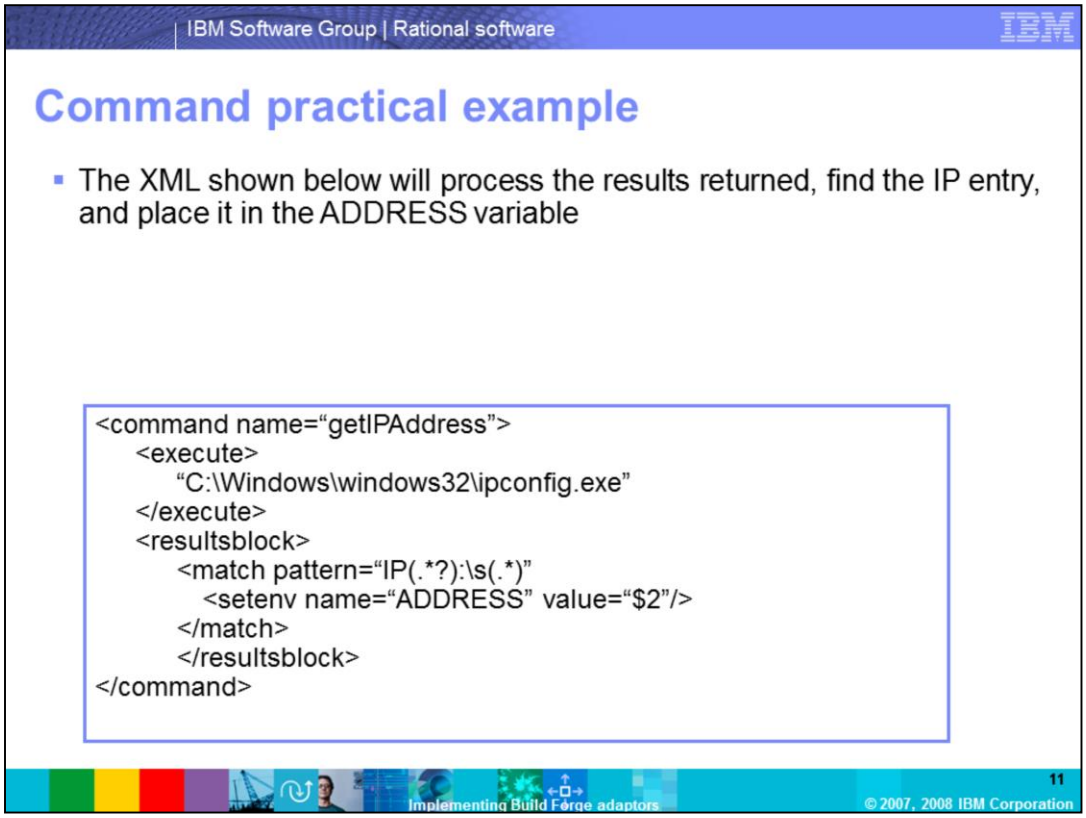
In version 7.0.1, the Command blocks are more complex. The command declaration can now define a *mode* as well as a *name*. The mode has three options: exec, conjoined, and parallel. Exec describes the commands as they worked before: execute as soon as you get them. Conjoined means that the commands will be collected together and executed as a batch. Last, parallel means that the commands will be threaded.



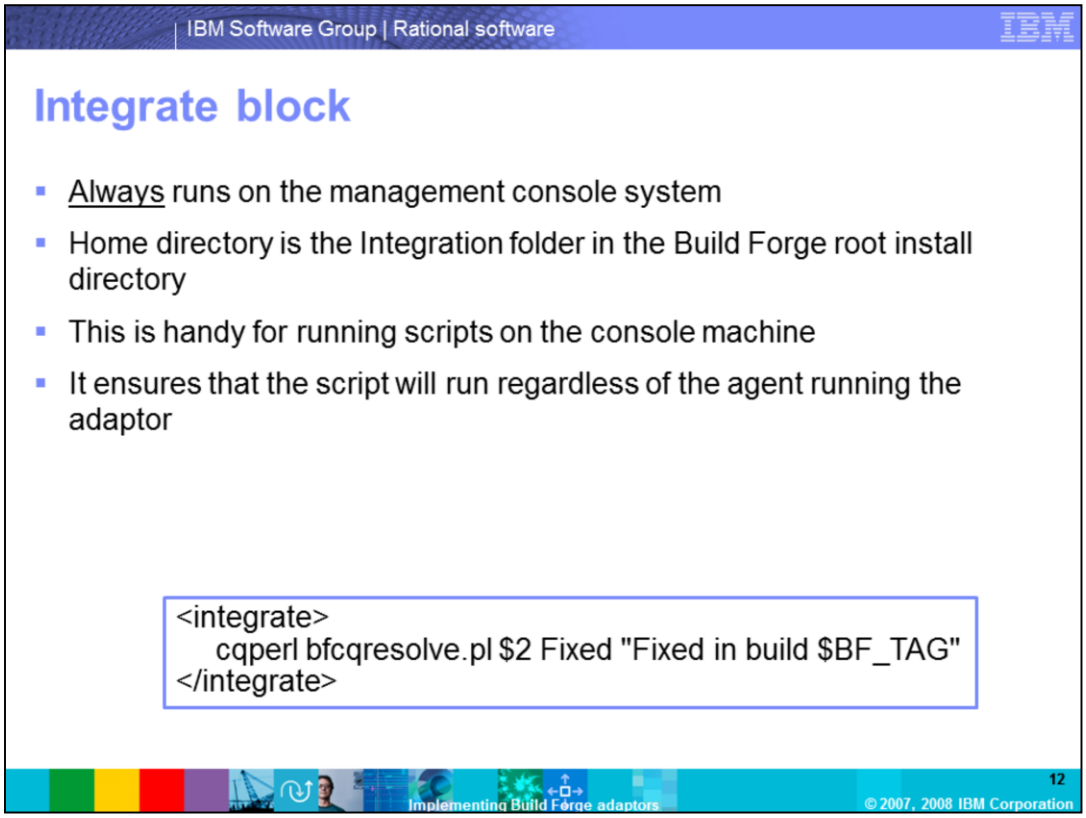
This is the Resultsblock section. The Resultsblock block takes the result from the command line execution, parses it, and then puts it back into Build Forge. This block is split up into one or more Match blocks. The Resultsblock block can optionally define a beginning and end pattern that is defined by a Perl regular expression. This might help you narrow down to a specific section to run Match blocks against. The Match blocks also define Perl regular expressions to parse and match sections of the return data. Note: Perl regular expressions will not be covered in this module.



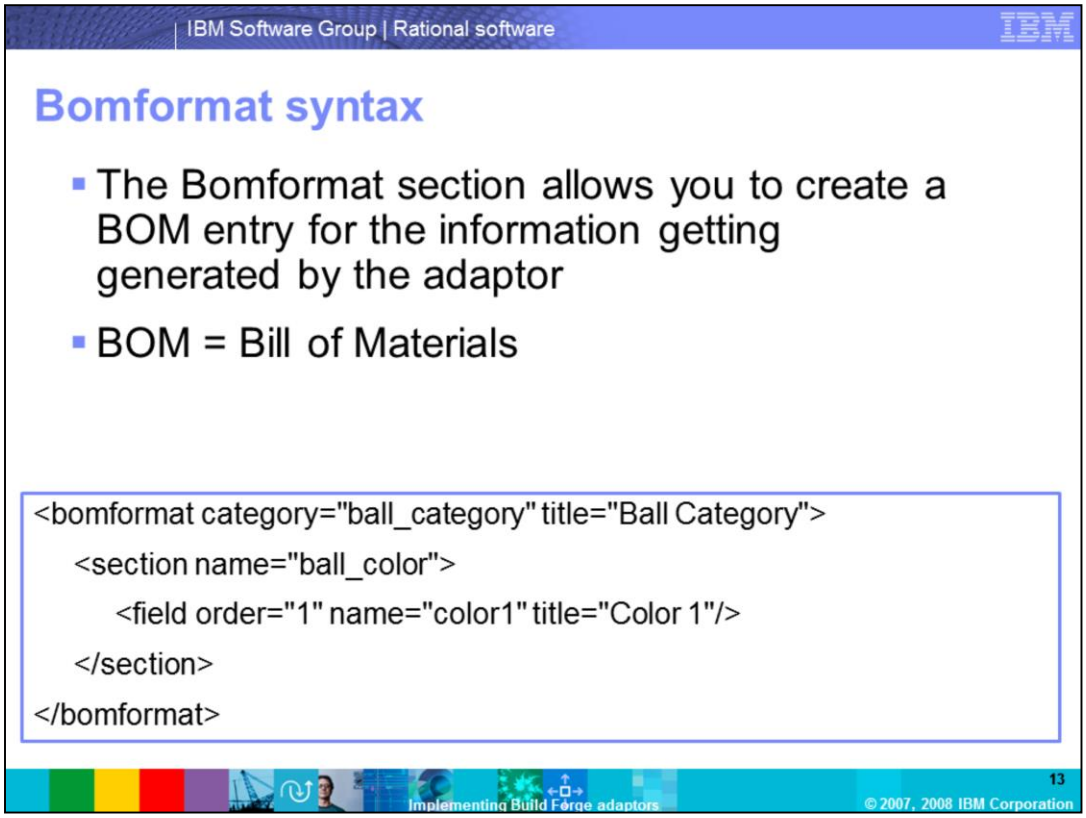
In this example, suppose you had a command that ran an *ipconfig* on a Windows system. From that command, you want to find the IP address and put it into a variable. Going back to the command block, you could put this common command in there. Here is an example of the ipconfig execution - so you know what sort of data you would get back. Now the question is: How do you use the Resultsblock to get useful information back from that output?



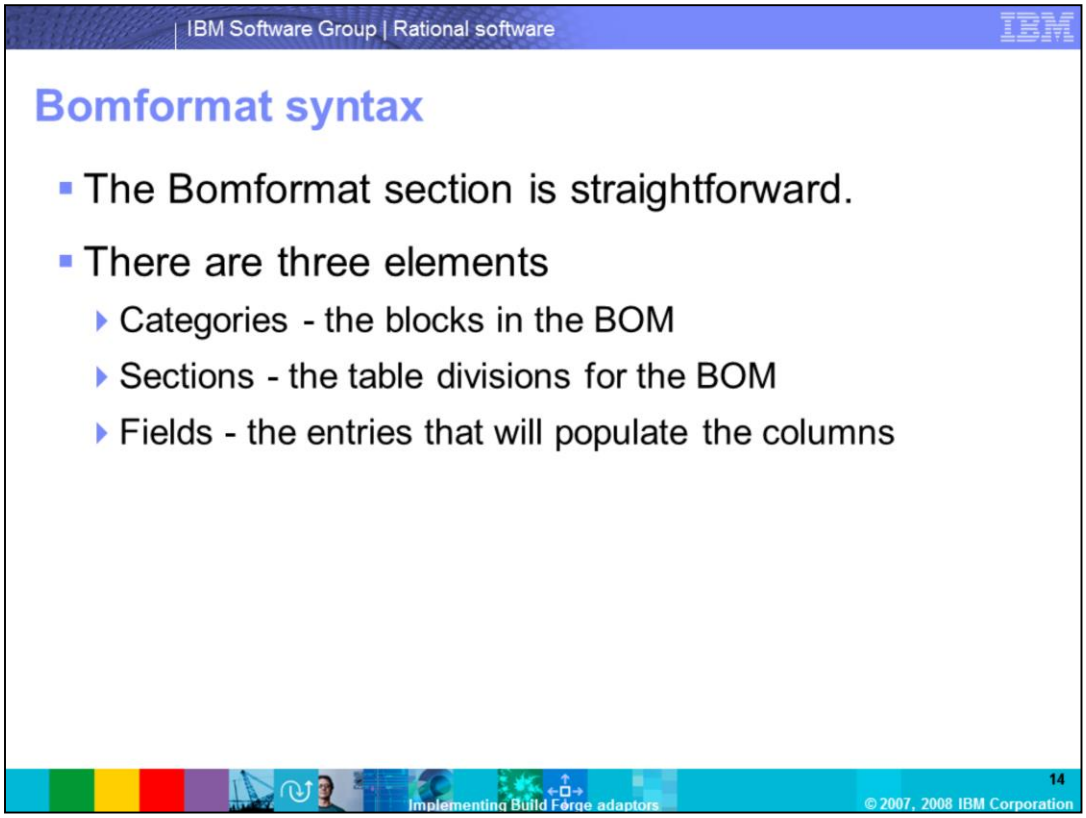
Here you can see the full implementation of the IPconfig example. Notice that you have set up the command to have a name of getIPAddress. You can then trace the flow of this as running the ipconfig.exe command, and the resulting information is sent to resultsblock. In this case there is a match block set up to capture the information that you are interested in. The match block uses a Perl regular expression to define the data that it is interested in. Perl regular expressions are beyond the scope of this presentation, but in this case you are looking for the IP address part of the output. Based on the regular expression matches you can then feed that into Build Forge. In this case you are setting the environment variable ADDRESS to $2 which is the part of the regular expression that was catching the IP address.



The integrate block is a special case replacement for the Execute block. When the command is called in the execute block, for example, a step, it is run on the agent defined by the run command. However in the case of Integrate, it will always run on the Management Console system from the home directory of Integration in the Build Forge root install. The intention for this was to make sure particular commands are consistently run on the same system (or if they are required to run on the console system).

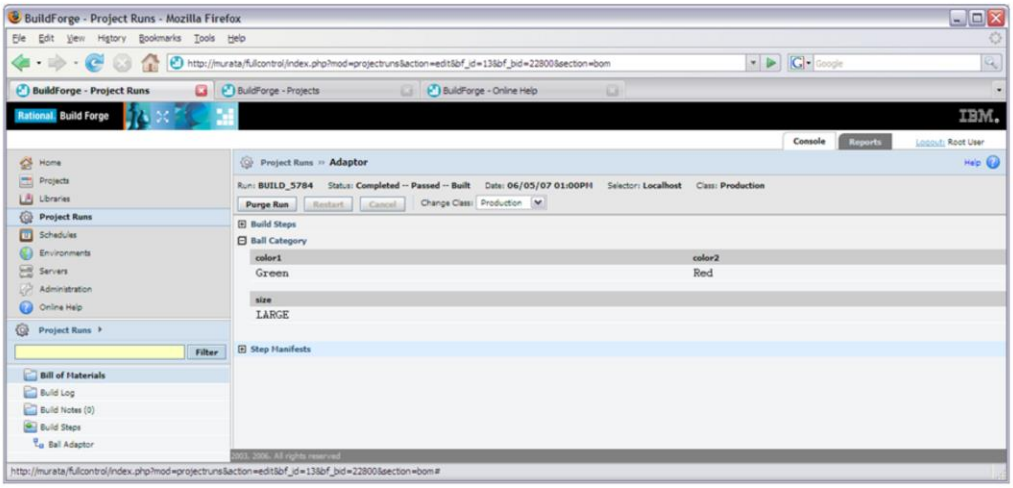


This is the Bomformat section. The Bomformat section allows you to create a BOM entry for the information getting generated by the adaptor. BOM stands for Bill of Materials.

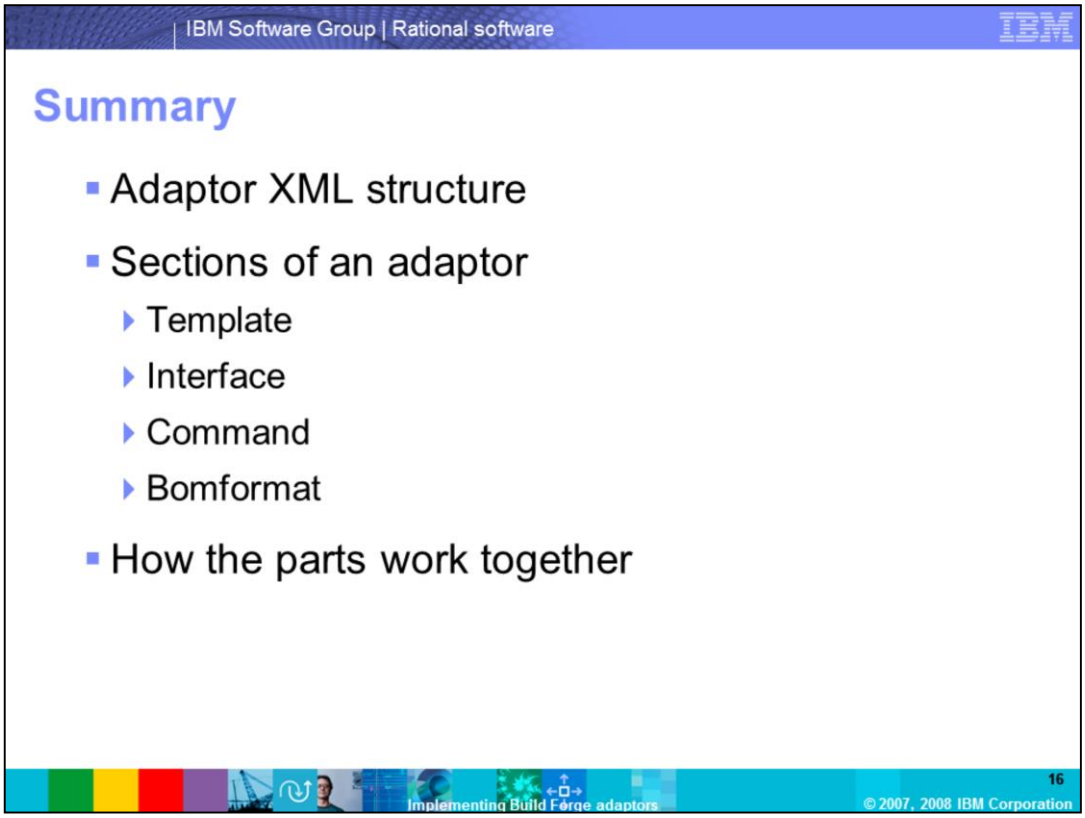


The Bomformat section is where you want to update the important parts of the build that you want to draw attention to (for example, what files were created by this build). With a regular build, those details are caught automatically, however with an Adaptor it is advantageous to add in your own details on what the Adaptor is doing and what it is touching. Anything from the command blocks can be captured and put into the BOM. The Bomformat block is simple, there are categories, sections, and fields. The categories are the expandable blocks that appear in the BOM. The sections are how the tables that appear in that block are split up. Finally the fields are the column entries in that table.

Bomformat continued



This screen capture shows you how the BOM appears for the example. You can see that the Category here is called “Ball Category”, with two sections; one of which has two fields called “color1” and “color2,” the other section has a field called “size.”



In summary, you should now be familiar with a simple adaptor. This module covered the Adaptor's XML structure, its sections in detail, including the template, interface, command, and bomformat syntax. In its entirety, you should now know how the parts of a Build Forge adaptor work together.

Trademarks, copyrights, and disclaimers