The slide features a blue background with a white grid pattern. At the top right is the IBM logo. Below it, the text 'IBM Software Group | Rational software' is displayed. The main title 'IBM Rational Build Forge V7.0.1' is in a large, bold, black font, followed by the subtitle 'Implementing Build Forge adaptors' in a smaller, italicized black font. A 'Rational software' logo is positioned below the subtitle. A horizontal bar with various icons (a person, a globe, a network diagram, etc.) spans the width of the slide. In the bottom right corner, the text '@business on demand.' is present, along with copyright information: '© 2007, 2008 IBM Corporation' and 'Converted to video June 29, 2015'.

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

Module objectives

- Topics covered in this section:
 - ▶ Adaptor XML structure
 - ▶ Sections of an adaptor
 - ▶ How the parts work together



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="300" />
  <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>
```

3

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.

Template syntax

- The template syntax is the first section of the adaptor. Its role is to define those variables that will be used by the adaptor.
- Those variables that are defined in the template block **MUST** be defined by the project calling the adaptor.
- The template block is the mechanism to make sure that those variables that will be used in the adaptor will exist.
- General rule: Any variable that the adaptor uses, regardless of scope, should be declared in the template block to prevent problems down the line.

```
<template>  
  <env name="COLOR" value="Red-Green" />  
  <env name="BALL_SERVER" value="ball" />  
</template>
```

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.

Interface syntax

- The Interface block is the second section of the adaptor
- The Interface block defines the entry point for Build Forge into the adaptor
- An adaptor definition can have many Interface blocks defined, but must have at least one
- The Interface block will define an executable method in the run that will kick off the rest of the adaptor; this is the entry for the adaptor

```
<interface name="ByColor">
  <run command="determine_color" params="$COLOR" server=$BALL_SERVER"
    dir="/" timeout="360" />
  <ontempenv name="Changes" value="empty">
    <step result="fail"/>
  </ontempenv>
</interface>
```

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.

Interface syntax

- The other important section is the ontempenv
- XML will define what the project should do after the adaptor
- Optionally, the interface will define what notifications will be made after the adaptor run

```
<interface name="ByColor">
  <run command="determine_color" params="$COLOR" server=$BALL_SERVER"
    dir="/" timeout="360" />
  <ontempenv name="Changes" value="empty">
    <step result="fail"/>
  </ontempenv>
</interface>
```

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.

Command syntax

```
<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>
```

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

Command syntax

- The command block defines the “methods” that the adaptor can use, and in turn the command block may call other commands
- The command block is split into two distinct parts
 - ▶ The execute block
 - ▶ The resultsblock block
- The first block is straightforward. The execution block defines the command to run on the command line

```
<execute>  
  cleartool -version  
</execute>
```

- This fulfills the first part of the adaptor; to run command line arguments, but how do you get the information back into Build Forge?



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.

Resultsblock syntax

- Takes the result from the command line execution, parses it, and then puts it back into Build Forge
- Split up into one or more match blocks
- Optionally define a beginning and end pattern that is defined by a Perl regular expression
- The match blocks define Perl regular expressions to parse and match sections of the return data

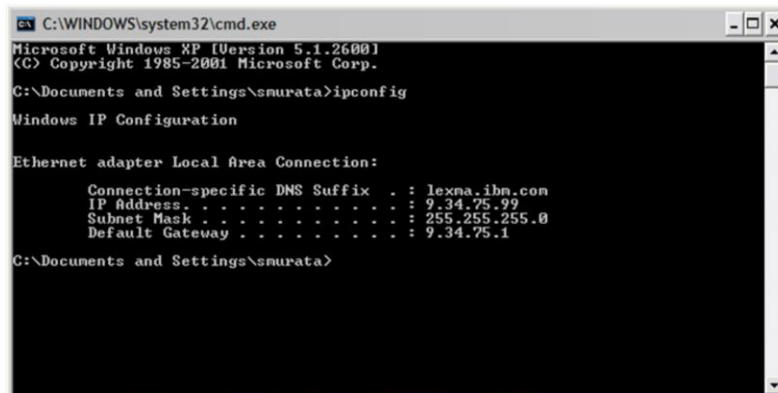
```
<resultsblock>
  <match pattern="^(.*?):(.*?)$">
    <bom category="ball_category" section="ball_color">
      <field name="color1" text="$1"/>
    </bom>
    <setenv group="Temp" name="COLOR" value="$2:$1"/>
    <setenv name="Changes" value="TRUE" type="temp append"/>
  </match>
```

9

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.

Resultsblock practical example

- Common command: ipconfig
- Take the results and put into a variable



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\smurata>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : lexna.ibm.com
    IP Address . . . . . : 9.34.75.99
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 9.34.75.1

C:\Documents and Settings\smurata>
```

10

Implementing Build Forge adaptors

© 2007, 2008 IBM Corporation

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?

Command practical example

- The XML shown below will process the results returned, find the IP entry, and place it in the ADDRESS variable

```
<command name="getIPAddress">
  <execute>
    "C:\Windows\windows32\ipconfig.exe"
  </execute>
  <resultsblock>
    <match pattern="IP(. *?):\s(.*)"
      <setenv name="ADDRESS" value="$2"/>
    </match>
  </resultsblock>
</command>
```

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.

Integrate block

- Always runs on the management console system
- Home directory is the Integration folder in the Build Forge root install directory
- This is handy for running scripts on the console machine
- It ensures that the script will run regardless of the agent running the adaptor

```
<integrate>  
  cqperl bfcqresolve.pl $2 Fixed "Fixed in build $BF_TAG"  
</integrate>
```



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).

Bomformat syntax

- The Bomformat section allows you to create a BOM entry for the information getting generated by the adaptor
- BOM = Bill of Materials

```
<bomformat category="ball_category" title="Ball Category">  
  <section name="ball_color">  
    <field order="1" name="color1" title="Color 1"/>  
  </section>  
</bomformat>
```

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.

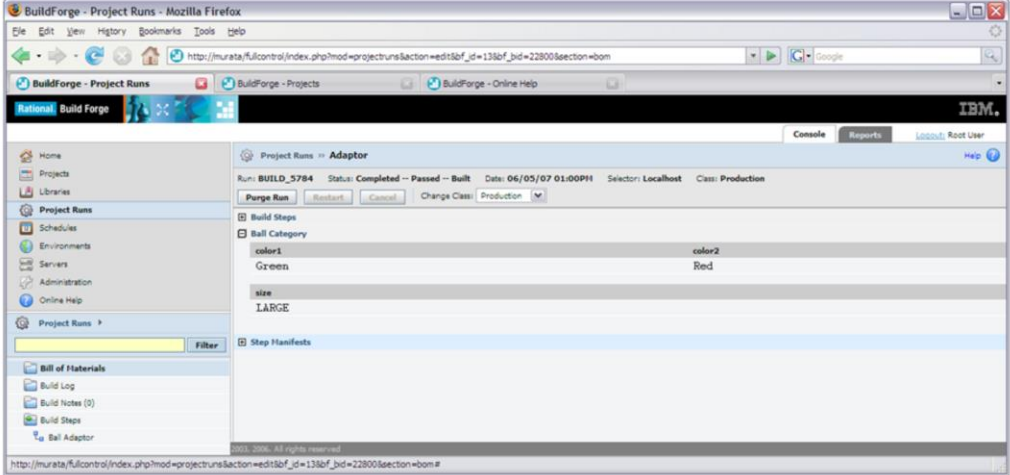
Bomformat syntax

- The Bomformat section is straightforward.
- There are three elements
 - ▶ Categories - the blocks in the BOM
 - ▶ Sections - the table divisions for the BOM
 - ▶ Fields - the entries that will populate the columns



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



The screenshot displays the BuildForge web interface in a Mozilla Firefox browser window. The page title is "BuildForge - Project Runs". The main content area shows a project run summary for "Adaptor" with the following details:

- Run: BUILD_5784
- Status: Completed -- Passed -- Built
- Date: 06/05/07 01:00PM
- Selector: Localhost
- Class: Production

Below the summary, there are buttons for "Purge Run", "Restart", and "Cancel", along with a "Change Class" dropdown menu set to "Production".

The "Build Steps" section is expanded, showing a "Ball Category" table with the following data:

color1	color2
Green	Red

Below the table, there is a "size" field with the value "LARGE".

The "Step Manifests" section is also visible but empty.

The footer of the screenshot includes the text "Implementing Build Forge adaptors" and "© 2007, 2008 IBM Corporation".

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.”

Summary

- Adaptor XML structure
- Sections of an adaptor
 - ▶ Template
 - ▶ Interface
 - ▶ Command
 - ▶ Bomformat
- How the parts work together



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

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

Build Forge IBM Rational

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, Other Countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. 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 in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

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

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2007. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.