

**IBM® Rational® Jazz®
Collaborative Lifecycle
Management**

**Process Enactment
Workshop for RTC
version 1.4**

Lab Exercises



Contents

LAB 1 INSTALL THE ENVIRONMENT	3
1.1 INSTALL OVERVIEW.....	4
1.2 WEB INSTALL.....	6
1.2.1 DOWNLOAD FILES	6
1.2.2 INSTALL THE SERVER	7
1.2.3 INSTALL THE CLIENT	10
1.3 INSTALLATION MANAGER INSTALL.....	11
1.3.1 DOWNLOAD FILES	11
1.3.2 INSTALL THE SERVER	12
1.3.3 INSTALL THE CLIENT	13
1.4 PLAIN ZIP INSTALL.....	15
1.4.1 DOWNLOAD FILES	15
1.4.2 INSTALL THE SERVER	15
1.4.3 INSTALL THE CLIENT	15
1.5 SUMMARY.....	17
1.6 APPENDIX A – INSTALL TIPS.....	18
LAB 2 SET UP THE ENVIRONMENT	20
2.1 SET UP THE SERVER ENVIRONMENT.....	21
2.1.1 CREATE AN ALIAS FOR THE HOST NAME	21
2.1.2 CONFIGURE TOMCAT PORTS	23
2.1.3 SERVER LANGUAGE	24
2.1.4 WEB UI LANGUAGE	25
2.2 SET UP THE RTC SERVER.....	26
2.3 SET UP THE RTC ECLIPSE CLIENT SOFTWARE.....	28
2.3.1 ECLIPSE CLIENT LANGUAGE	28
2.3.2 ECLIPSE CLIENT USER PREFERENCES	28
2.4 CREATE A TEST PROJECT.....	31
2.5 SUMMARY.....	33
2.6 APPENDIX A – ADDITIONAL CONSIDERATIONS.....	34
LAB 3 CONFIGURING WORK ITEMS	35
3.1 CUSTOMIZE THE PROCESS IN RTC.....	36
3.1.1 UNDERSTAND THE NEW NEEDS.....	36
3.1.2 CREATE A NEW WORK ITEM TYPE WITH CUSTOM ATTRIBUTES.....	39
3.1.3 CREATE A WORKFLOW.....	44
3.1.4 CUSTOMIZE THE PRESENTATION FOR THE WORK ITEM TYPE.....	48
3.1.5 CONFIGURE PERMISSIONS	53
3.2 GATHER THE PROCESS CHANGES.....	55
3.3 SUMMARY.....	58
LAB 4 WORK ITEM CUSTOMIZATION	59
4.1 DEFAULT VALUE PROVIDER.....	61
4.1.1 DEFAULT VALUES FOR ATTRIBUTES OF TYPE CONTRIBUTOR	61
4.1.2 ROLE BASED ENUMERATION DEFAULT	67
4.1.3 OPTIONAL: REVIEW THE OTHER AVAILABLE DEFAULT VALUE PROVIDERS	73

4.1.4 SUMMARY	74
4.2 VALUE SETS.....	76
4.2.1 DEPENDENT ENUMERATIONS	76
4.2.2 HTTP FILTERED VALUE SET	82
4.2.3 REFINE THE HTTP FILTERED VALUE SET	93
4.2.4 ADDITIONAL INFORMATION FOR HTTP BASED FILTERED VALUE PROVIDERS	97
4.2.5 ROLE BASED USER LIST	98
4.2.6 SUMMARY	102
4.3 VALIDATORS.....	103
4.3.1 SUMMARY	112
4.4 SUMMARY.....	113
LAB 5 WORK ITEM CUSTOMIZATION WITH JAVASCRIPT	114
5.1 INTRODUCTION TO JAVASCRIPT BASED ATTRIBUTE CUSTOMIZATION.....	115
5.1.1 JAVASCRIPT BASED ATTRIBUTE CUSTOMIZATION CAPABILITIES AND LIMITATIONS	117
5.1.2 CHALLENGES DEVELOPING JAVASCRIPT BASED ATTRIBUTE CUSTOMIZATION	119
5.1.3 ENABLE JAVASCRIPT	120
5.1.4 OPTIONAL: INSTALL THE WEB, XML, AND JAVA EE DEVELOPMENT TOOLS	121
5.2 SCRIPT BASED CALCULATED VALUE.....	124
5.2.1 TOTAL COST CALCULATED VALUE	124
5.2.2 “ATTRIBUTEVALUEANALYZER” CALCULATED VALUE PROVIDER	145
5.2.3 SUMMARY	153
5.3 SCRIPT BASED CONDITIONS.....	154
5.4 SCRIPT BASED VALIDATIONS.....	167
5.5 SCRIPT BASED VALUE SET.....	175
5.6 CALCULATED VALUE TO VISUALIZE THE STATE OF THE TECHNOLOGY REVIEW.....	181
5.7 SUMMARY.....	190
5.8 SOLUTIONS.....	191
5.9 APPENDIX A - SCRIPT TROUBLESHOOTING.....	192
5.10 APPENDIX B – SCRIPT DEBUGGING.....	193
5.10.1 SCRIPT DEBUGGING EXAMPLE WITH CHROME	193
5.10.2 SCRIPT DEBUGGING EXAMPLE WITH FIREBUG	193
5.11 APPENDIX C – SCRIPTED HTTP VALUE SET PROVIDER.....	198
5.12 APPENDIX D – CHANGES IN 4.0.3 ATTRIBUTE CUSTOMIZATION SCRIPT BASED EDITOR.....	204
APPENDIX A GLOSSARY	207
APPENDIX B NOTICES	209
APPENDIX C TRADEMARKS AND COPYRIGHTS	211

Lab 1 Install the Environment

In this lab you will install your environment. This involves installing client and server software.

Process development should occur in a separate Rational Team Concert (RTC) web application from the regular production server. RTC projects can't be deleted at this time, so developing them in a separate RTC application keeps the production environment cleaner. It's also good to develop and test processes in a separate RTC application so you don't have to worry about accidentally changing settings in the version of RTC that everyone else is using. You'll be free to experiment and create without impacting other team members.

You will install the following software in this lab:

- The server: RTC with JTS, used for process development.

For the purposes of the workshop, these do not represent the real-life JTS and RTC web applications your organization uses to manage projects, work items, etc. They are solely used for process development.

- An RTC client (Eclipse).

When you're through with this lab, you will have learned:

- How to install all the software

Why a separate server for process development?

There are several reasons for this.

It is currently not possible to delete projects on an RTC server. You can only archive projects.



During process development you will experiment with several potential solution approaches. Since some decisions can not be changed after they have been used, for example work item attribute types, you might have to create several projects until you finally found the right implementation.

On the production server you might not have all the permissions.

Therefore the separate process development server allows for experimenting and try different solution approaches without impacting the work on the production server.

1.1 Install Overview

The description below suggests a special folder for the test system. You should use the recommended location. If this is not possible, replace paths in the instructions pointing to that location with your install path.

If you want to set up Rational Team Concert, you have basically three options to do this.

- 1) You can use the Web Installer to download and install all the required software
- 2) You can download IBM Installation Manager repositories and use them to install the required software
- 3) You can download the software as “Plain ZIP Install”, using compressed ZIP files; and decompress the files to install all the required software

Which option should be used when?



Use The Plain ZIP Install

The Plain ZIP install is the easiest option available for this workshop as the other CLM tools are not needed.

Use the Web Install if you

- Want to do a one time only install
- Have a reliable network connection with a high bandwidth
- Want to install more than Just Rational Team Concert
- Have not yet any IBM Installation Manager or ZIP install files available

Use the Installation Manager Repository Install if you

- Want to install Rational Team Concert or any other CLM tool like Rational Quality Manager multiple times
- Want to do a distributed install on separate servers
- Have to provide the tools to many users
- Have issues with your internet connection (use a download manager to download the repositories)

Use the Plain ZIP Install if you

- Only need to install RTC, the Eclipse client or the Buildsystem Toolkit – the other applications are not available as ZIP only install

- Want small footprint

The following sections describe each option.

Installation instructions and versions

The installation instructions in this lab describe the basic steps with the purpose of serving as guideline. Installation in some environments may require additional or different steps (e.g. RTC client integration with other tools).



The described installation procedures will also use CLM version 4.0.0.1, which was the version used at the time this workshop was first developed. However, as general rule, you will download and install the latest available product released version.

There might be small differences in the screen shots and small differences in the process for newer versions of RTC. If in doubt check the documentation of your version of RTC.


1.2 Web Install

1.2.1 Download Files

- __1. Create a directory in the root folder of your local disk called *C:\JSWorkshops*. Create a sub-directory named *Downloads*. You will download all files into this directory.
- __2. Download and install Firefox <http://www.mozilla.org/en-US/firefox/organizations/all.html>
 - __a. Select the **Firefox > Options** and change the following settings:
 - __i. General tab
 - __a. Save files to *C:\JSWorkshops\Downloads*
 - __ii. Content tab
 - __a. Turn off “pop-up blocking”. CLM applications sometimes use pop-ups for logging in.
 - __iii. Advanced
 - __a. Select **Never check for updates**. As of this writing, Firefox 10 is the version supported by CLM, so you should keep it at that level.
- __3. Download CLM 4.0.x
 - __a. Go to the [All Downloads tab on the CLM 4.0.0.1 download page](#) (or the page for the latest version of CLM).
- __4. In the Web Installers section, download the zip file for Windows x86 (or the platform you're using), into *C:\JSWorkshops\Downloads* folder

1.2.2 Install the Server

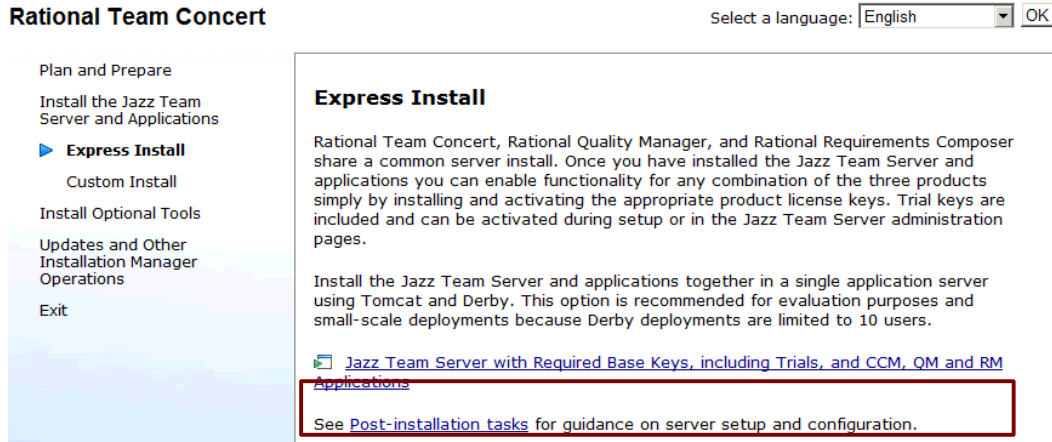
- __1. Install the JTS and RTC applications.
 - __a. Use [7Zip](#) (or the application of your choice) to unzip the downloaded CLM web installer into a sub-directory of the *Downloads* folder.
 - __b. In the chosen sub-directory execute **launchpad.exe**.
 - __c. In the first window, select the link **Install the Jazz Team Server and Applications**. Then select **Express Install**.



Why Express Install?

We are doing a simple installation with all the products collocated in the same server using Tomcat and Derby. You can select Custom installation for flexible installation options of the applications.

- __d. In the installation window, select the link **Jazz Team Server with Required Base Keys, including Trials, and CCM, QM and RM Applications**.



Rational Team Concert Select a language:

Plan and Prepare

Install the Jazz Team Server and Applications

▶ **Express Install**

Custom Install

Install Optional Tools


Updates and Other Installation Manager Operations

Exit

Express Install

Rational Team Concert, Rational Quality Manager, and Rational Requirements Composer share a common server install. Once you have installed the Jazz Team Server and applications you can enable functionality for any combination of the three products simply by installing and activating the appropriate product license keys. Trial keys are included and can be activated during setup or in the Jazz Team Server administration pages.

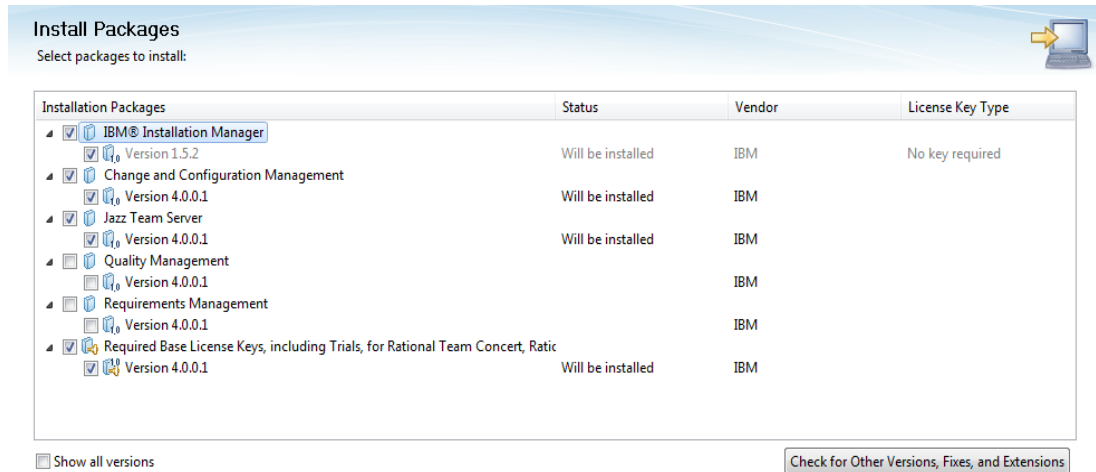
Install the Jazz Team Server and applications together in a single application server using Tomcat and Derby. This option is recommended for evaluation purposes and small-scale deployments because Derby deployments are limited to 10 users.

 [Jazz Team Server with Required Base Keys, including Trials, and CCM, QM and RM Applications](#)


See [Post-installation tasks](#) for guidance on server setup and configuration.

- __e. Log in to *Jazz.net* using your regular *Jazz.net* registered user credentials if asked.

- __f. In the first window, select the following packages: “Installation Manager”, “Change and Configuration Management”, “Jazz Team Server” and “license keys”.




Why that packages?



In this lab we are focused on Rational Team Concert installation, and those are the required packages for installing RTC server. You can optionally install Rational Quality Manager and Rational Requirements Composer selecting their installation packages as well.

- __g. Select **Next**.

Windows 7 Supported OS Warning



If you are running under Windows 7 you may see a Supported Operating System warning. You can ignore this and select Next.

- __h. Read the license agreement and if you accept, select **Next**.
- __i. If you installation manager the first time on this machine, change the Shared Resources Directory to *C:\JSWorkshops\IBM\IBMIMShared*, and change the Installation Manager directory to *C:\JSWorkshops\IBM\Installation Manager\eclipse*. Select **Next**.

Install Packages
Select a location for the shared resources directory and a location for Installation Manager.

Install Prerequisite Licenses **Location** Features Summary

When you install packages, files are stored in two locations:

- 1) The shared resources directory - resources that can be shared by multiple packages.
- 2) The installation directory - any resources that are unique to the package that you are installing.

Important: You can only select the shared resources directory the first time you install a package with the IBM Installation Manager. For best results select the drive with the most available space because it must have adequate space for the shared resources of future packages.

Shared Resources Directory: C:\JSWorkshops\IBM\IBMIMShared

Once installed, IBM Installation Manager will be used to install, update, modify, manage and uninstall your packages.

Installation Manager Directory: C:\JSWorkshops\IBM\Installation Manager\eclipse

- __j. Change the installation directory to *C:\JSWprkshops\IBM\JazzTeamServer*. If you're running on a 64-bit system, make sure the *Architecture Selection* is 64-bit. Select **Next**.
- __k. Select any additional languages, then select **Next**.
- __l. Verify that you are installing the correct applications, then select **Next**.

Install Packages
Select the features to install.

Install Prerequisite Licenses Location **Features** Summary

Features

- Change and Configuration Management 4.0.0.1
- Jazz Team Server 4.0.0.1
- Required Base License Keys, including Trials, for Rational Team Concert, Ra
- IBM® Installation Manager 1.5.2

- __m. Make sure **Install Tomcat 7** is selected under *Web Application Location*. Select **Next**.
- __n. Make sure **Use default 3.x/4.x application context roots** is selected under *Context Root Options*. Select **Next**.
- __o. Verify the installation options, then select **Install**.
- __p. On the final page displayed when installation is complete, select **None**, then **Finish**.

1.2.3 Install the Client

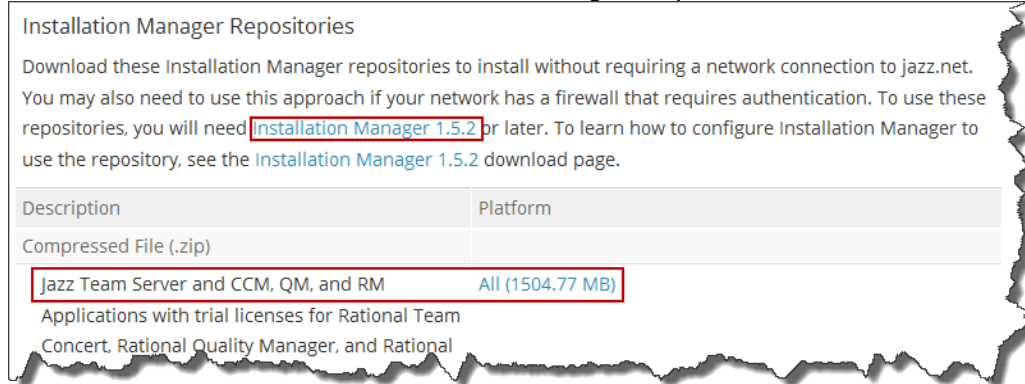
- __1. Install the Rational Team Concert Eclipse client
 - __a. Return to the **Launchpad** window (restart launchpad.exe if it is not open) and under *Install Optional Tools* select the link for **Rational Team Concert – Client for Eclipse IDE. Installation Manager** will launch.
 - __b. Log in to *Jazz.net* using your regular *Jazz.net* registered user credentials if asked.
 - __c. On the first screen, select **Rational Team Concert – Client for Eclipse**. Make sure **Version 4.0.0.1** is also checked (or the latest version), then select **Next**.
 - __d. Read the license agreement and if you accept it, select **Next**.
 - __e. Change the installation directory to <C:\JSWorkshops\IBM\TeamConcert>.
 - __f. Under *Architecture Selection*, select **64-bit**. Select **Next**.
 - __g. Do **NOT** check **Extend an existing Eclipse**. Select **Next**.
 - __h. Select any additional languages, then select **Next**.
 - __i. In **Select the features to install**, make sure **Rational Team Concert – Client for Eclipse IDE 4.0.0.1** is selected. Do NOT select **Sametime Integration Update Site**. Select **Next**.
 - __j. Under Common Configurations, select **Access help from the Web**. Select **Next**.
 - __k. On the review page, make sure the installation directories and packages are correct, and select **Install**.
- __2. After the installation completes, close the Launchpad.

1.3 Installation Manager Install

1.3.1 Download Files

__1. Go to the [All Downloads tab on the CLM 4.0.0.1 download page](#) (or the [page for the latest version of CLM](#)).

__a. Scroll Down to the section “Installation Manager Repositories”



__b. If it is not already installed, download IBM “Installation Manager” to *C:\JSWorkshops\Downloads* and install it

__i. Use *C:\JSWorkshops\IBM\Installation Manager* as install location

__ii. Use *C:\JSWorkshops\IBM\IBMIMShared* as shared location if asked

__c. Download the “Jazz Team Server and CCM, QM, and RM Applications with Trial licenses for Rational Team Concert, Rational Quality Manager, and Rational Requirements Composer” Installation Manager Repository to *C:\JSWorkshops\Downloads*

__i. Use [7Zip](#) (or the application of your choice), to decompress the downloaded repository into a sub folder.

__d. Download the “Client for Eclipse IDE” Installation Manager Repository to *C:\JSWorkshops\Downloads*

__i. Use [7Zip](#) (or the application of your choice), to decompress the downloaded repository into a sub folder.

__2. Start Installation Manager and use the **File > Preferences** menu and add the repositories:

__a. Navigate to the Repositories menu. Repeat the following steps two times, for two downloaded repositories:

__i. Select **Add Repository...**

- __ii. Select **Browse** and navigate to the location where you unzipped the downloaded repository. Select **OK**.
- __b. Select **OK**.
- __c. Make sure you **uncheck** Search service repositories during installation and updates.

Service repositories are remote locations where updates or extensions to packages (including the Installation Manager itself) are stored.

Search service repositories during installation and updates.

- __d. Select **OK**.

1.3.2 Install the Server

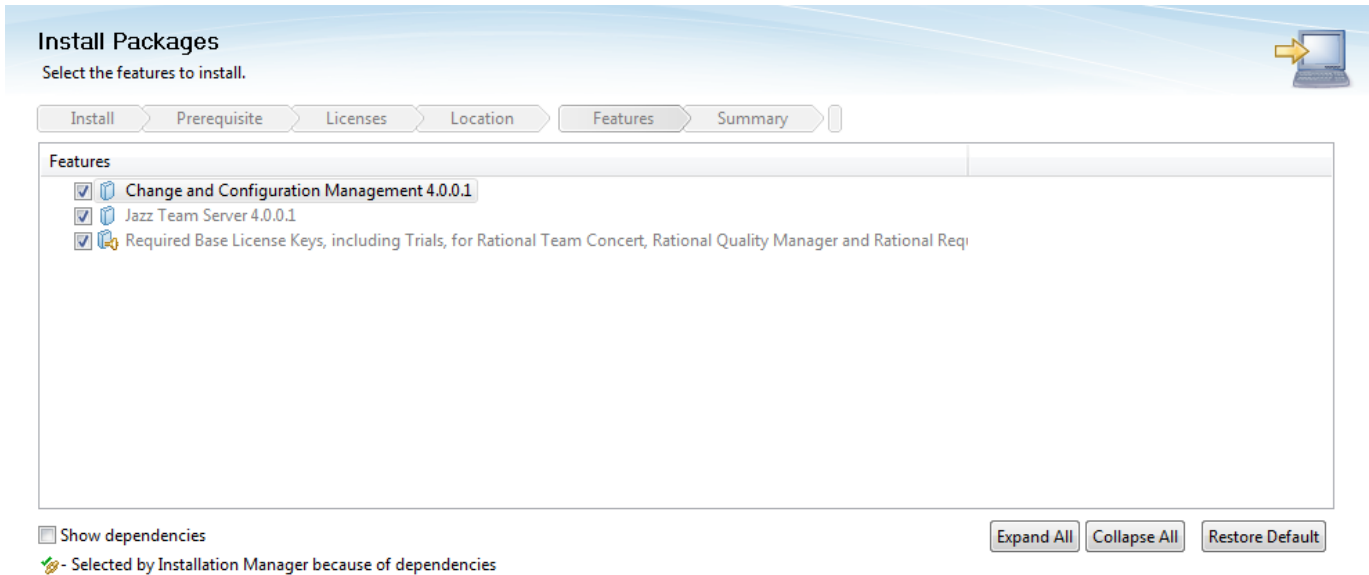
- __1. On the Installation Manager Main page select **Install**.
 - __a. On the first page of the install, select “Change And Configuration Management” and “Jazz Team Server”. You can select “Requirements Management” and “Quality Management” also to install these applications, but they are not required.
 - __b. Select **Next**.
 - __c. Read the license agreement and if you accept, select **Next**.



Windows 7 Supported OS Warning

If you are running under Windows 7 you may see a Supported Operating System warning. You can ignore this and select Next.

- __d. Change the Shared Resources Directory to *C:\IBM\IBMIMShared* or *C:\JSWorkshops\IBM\IBMIMShared*. Select **Next**.
- __e. Change the installation directory to *C:\JSWorkshops\IBM\JazzTeamServer*.
- __f. If you're running on a 64-bit system, make sure the *Architecture Selection* is 64-bit. Select **Next**.
- __g. Select any additional languages you want to install, then select **Next**.
- __h. Verify that you are installing the correct applications, then select **Next**.



- __i. Make sure **Install Tomcat 7** is selected under *Web Application Location*. Make sure **Use default 3.x/4.x application context roots** is selected under *Context Root Options*. Select **Next**.
- __j. Verify the installation options, then select **Install**.
- __k. On the final page displayed when installation is complete, select **None**, then **Finish**.

1.3.3 Install the Client

Install the Rational Team Concert Eclipse client:

- __1. Return to the Installation Manager Main window (restart Installation Manager if it is not open) and select **Install**.
 - __a. Under *Install Optional Tools* select the link for **Rational Team Concert – Client for Eclipse IDE** then select **Next**.
 - __b. Read the license agreement and if you accept it, select **Next**.
 - __c. Change the installation directory to <C:\JSWorkshops\BM\TeamConcert>.
 - __d. Under *Architecture Selection*, select **64-bit**. Select **Next**.
 - __e. Do **NOT** check **Extend an existing Eclipse**. Select **Next**.
 - __f. Select any additional languages, then select **Next**.
 - __g. In **Select the features to install**, make sure **Rational Team Concert – Client for Eclipse IDE 4.0.0.1** is selected. Do NOT select **Sametime Integration Update Site**. Select **Next**.

- ___h. Under Common Configurations, select **Access help from the Web**. Select **Next**.
- ___2. On the review page, make sure the installation directories and packages are correct, and select **Install**.

1.4 Plain ZIP Install

1.4.1 Download Files

- ___1. Go to the [All Downloads tab on the CLM 4.0.0.1 download page](#) (or the [page for the latest version of CLM](#)).
- ___a. Scroll Down to the section “Plain .zip Files”.

Plain .zip Files

Extract these .zip files to quickly install specific IDE-based clients or other tools. The Client for Microsoft Visual Studio IDE can only be installed by using the web installer or by installing locally using the IBM Installation Manager repository. Compressed (.zip) files are used for installing the server and license keys on z/OS and IBM i platforms for milestone builds. LICPGM (IBM i) and SMP/E (zOS) packages will be available for the final release.

Description	Platform
Compressed File (.zip)	
Jazz Team Server and the CCM Application, and Trial licenses for Rational Team Concert	Windows x86 (626.54 MB)
	Windows x86-64 (639.32 MB)
	Linux x86 (612.46 MB)
	Linux x86-64 (618.73 MB)
Client for Eclipse 3.6.x IDE	Windows x86 (416.21 MB)
	Linux x86 (403.77 MB)

- ___b. In the Plain .zip Files section, download the compressed “Jazz Team Server and the CCM Application, and Trial licenses for Rational Team Concert” file for Windows x86 (or the platform you're using) to *C:\JSWorkshops\Downloads*. The File is most likely named *JTS-CCM-keys.XXXX_YYYY* with XXXX representing the architecture and YYYY representing the version.
- ___c. In the Plain .zip Files section, download the compressed “Client for Eclipse IDE” file for Windows x86 (or the platform you're using) to *C:\JSWorkshops\Downloads*. The File is most likely named *RTC-Client-XXXX_YYYY* with XXXX representing the architecture and YYYY representing the version.

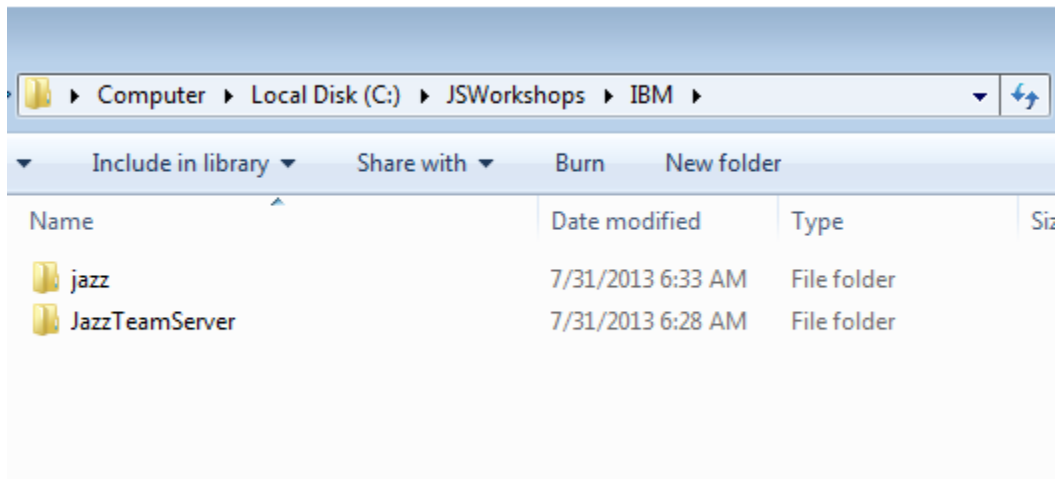
1.4.2 Install the Server

- ___1. Select the *JTS-CCM-keys-XXXX_YYYY.zip* and use [7Zip](#) to extract the content to **C:\JSWorkshops\IBM\JazzTeamServer**
The server will be ready to be used once the package gets uninstalled.

1.4.3 Install the Client

- ___1. Select the *RTC-Client-XXXX_YYYY.zip* and use [7Zip](#) to extract the content to **C:\JSWorkshops\IBM**
The client will be ready to be used once the package gets uninstalled.

__2. Your folder C:\JSWorkshops\IBM should now look like below



1.5 Summary

Congratulations! You've installed your process development environment. You now have the following in your process development environment:

- A RTC Server for process development with a JTS and associated CCM application. This is where you'll develop your new process templates and descriptions to simulate how a real project would make use of them.
- The Rational Team Concert Eclipse client.

1.6 Appendix A – Install Tips

This Appendix shares some best practices identified over the time when installing CLM servers such as RTC.

- **On windows, avoid installing into Program Files**

These folders are protected by Windows even if logged in as administrator and prevent from creating folders in the substructure. During start up, the Jazz Servers need to create folders and unpack files in the folder substructure. Since this is prevented by the OS the start up will fail, unless you explicitly run the server start up using Run As Administrator.

- **Install the Installation Manager and shared resources into one folder e.g. /IBM or /opt/ibm**

This has proven to be a best practice, because it is a one time choice on a machine and can not easily be changed later. To change it you would have to uninstall Installation Manager and everything installed using it first.

Using a dedicated folder allows to easily find the location, especially if you make it a general rule for all your installs. It will help you in case it is necessary to deploy hot-fixes for shared components.

A typical example would be using the folders below:

- /IBM/Installation Manager
- /IBM/IBMIMShared

- **Install the CLM Products into a short path, in a production Install consider to provide a version number in the path**

A dedicated location helps finding the location e.g. for back up. It makes running and marinating different versions on a machine easier.

The Eclipse and Java Package naming conventions create long path names that can create issues on some operating systems.

The version number shows in Installation Manager dialogs helping with identifying what you deinstall. It also helps with upgrades, especially running upgrade scripts. The version number is clearly visible in provided parameters and errors are more likely to be detected.

It is easier to understand which files can be deleted after an upgrade.

A typical setup is presented below

- /clm/4.0/JazzTeamServer
- /clm/4.0.0.1/JazzTeamServer

- **Change the [Index File](#) locations to a well known location with high bandwidth, enough disk space and specify absolute paths for the index files in the `teamserv.properties`**

The [Index Files](#) are used by the applications in several search operations, high bandwidth avoids adverse effects on performance.

The [Index Files](#) partake in upgrade operations. They are upgraded. Using relative paths can cause issues when upgrading, requiring a copy of the files or a loss of data.

The [Index Files](#) need to be included in backup operations as described [here in the Deployment Wiki](#). The backup procedure might fail for the [Index Files](#) after an upgrade, or restore.

Using the default locations in the server install folder with an absolute path can cause a loss of the [Index Files](#) when deleting files of an older install after an upgrade.

See a possible choice of the location for the index files below

- `/clm/index`

- **Temporary Files**

It is not well known that the Jazz Products also use some temporary files to store data. The temporary files require space and permission to store these files. On windows you can find the files by default in as the folders [C:/tmp/contentservice](#) and [c:/tmp/versionedcontentservice](#).

If you accidentally delete those folders and the server gets confused, you need to restart the server.

These locations can be modified in the “Advanced Properties” of the Server Administration page, of your Jazz product.

- **Temporary Files and Linux machines: if you plan to run the server with non-root user**

Running the server under a user account different from root (and without sudo), requires that you make sure that the user ID you plan to use have write permission to the “/tmp” location, and specially the temporary folders that Jazz Products use, by default `/tmp/contentservice` and `/tmp/versionedcontentservice`. It is worth checking this permissions before first execution as they are usually constrained in a Linux environment.

Lab 2 Set Up the Environment

In this lab you will set up your process development environment. This involves setting up the client and server software and creating a project area.

You will set up the following software in this lab:

- JTS and RTC applications
- A RTC client (Eclipse) and the required workspace

When you're through with this lab, you will have learned:

- How to set up the server
- The importance of IP ports
- How to set up the RTC Eclipse Client
- How to create a project from a template
- How to assign Roles to users

2.1 Set Up The Server Environment

Every Jazz Server requires a Public URI. This URI is used to access the server, and it's also used to generate the references to the repository resources for accessing them, and to link data between applications.



Public URI and Test Environment

Planning the Public URI you will use in your CLM installation is one of the most important steps in your deployment planning. See [Planning your URIs](#).

While for a Test Environment it won't generally be that important as for a Production one, a minimum decision regarding the URI so it is meaningful and allows you to take the most of your Test Environment is advised.

When defining the Public URI for your deployment, “localhost” is frowned upon. Although this does not matter much for a test system, the suggestion is to avoid using it.

In addition it is important to know how to change the port for the server, in case you want to run several servers on one machine, or comply with some standards in your organization.

The next section shows how this can be done.

2.1.1 Create an Alias for the host name

__1. To be able to use fully qualified domain names without actually having to change the domain name server, use a host Alias.

__a. Open the hosts file for editing as an administrator by right-clicking Notepad.exe and clicking **Run as Administrator**. On Windows, the hosts file is located at *C:\Windows\System32\drivers\etc\hosts*.

__b. Add the following line to the bottom of the file: **127.0.0.1 clm.process.ws**

```
# localhost name resolution is handled within DNS itself.
127.0.0.1    localhost
127.0.0.1    clm.process.ws
```

__c. **Save** and close the file.

What Did I Just Do?

By adding that line to the hosts file, you have told your networking software that any requests to a machine named `clm.process.ws` should be routed to the `127.0.0.1` address (which is your local machine).



If you were doing this in a REAL environment, you would have your systems administrators make an entry into the DNS tables (which are used to resolve host names) that would route the base of your selected Public URI to the machine hosting your Jazz infrastructure.

2.1.2 Configure Tomcat Ports

- __1. Configure Tomcat to serve applications on HTTP and HTTPS well known default ports:

We're doing this because...

We have decided that our public URI for CLM deployment is going to be *clm.process.ws*. We want it to be served with no reference to ports on the server to make user access easier.



This will also allow our installation to scale in the future and leverage middleware like Web Server or DNS virtual names. We won't be scaling to large topologies in this workshop, but it's a good practice to install this way when doing evaluation topologies. It will make future changes easier if your topology needs to evolve into a more robust environment.

- __a. Open the server configuration file
C:\JSWorkshops\IBM\JazzTeamServer\server\tomcat\conf\server.xml for editing.
- __i. Look for the string '9443'
- __ii. Change all the occurrences in non-commented nodes of that *port* and *redirectPort* attribute value to **443**
- __iii. Perform another search looking for the string '9080'
- __iv. Change the occurrences in non-commented nodes of that *port* attribute value to **80**

Information



Just the Connector nodes for HTTP and HTTPS are required to be changed for CLM. However, you don't want redirection to a port you don't want to use in your deployment and which may be in use by other services.

The resulting nodes for HTTP and HTTPS will look like the following:

...

```
<Connector port="80" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="443" />
```

...


```

<Connector port="443"
    connectionTimeout="20000"
    maxHttpHeaderSize="8192"
    maxThreads="150"
    minSpareThreads="25"
    enableLookups="false"
    disableUploadTimeout="true"
    acceptCount="100"
    scheme="https"
    secure="true"
    clientAuth="false"
    keystoreFile="ibm-team-ssl.keystore"
    keystorePass="ibm-team"
    protocol="HTTP/1.1"
    SSLEnabled="true"
    sslProtocol="{jazz.connector.sslProtocol}"
    algorithm="{jazz.connector.algorithm}"
    URIEncoding="UTF-8" />

```

__b. **Save** your changes and exit.

2.1.3 Server Language

The Jazz server picks up its language based on the language settings of the server. This can result in a mix of languages. Perform the following steps if you want to enforce a server language different than the one set on a server.

__1. Modify server startup language setting

__a. Open `C:\JSWorkshops\IBM\JazzTeamServer\server\server.startup` with an editor

__b. Add `-Duser.language=en` to the end of the `JAVA_OPTS` in `C:\JSWorkshops\IBM\JazzTeamServer\server\server.startup`. The file should now look as below:

```

set JAVA_OPTS=%JAVA_OPTS% -Xcompressedrefs -Xgc:preferredHeapBase=0x100000000
set JAVA_OPTS=%JAVA_OPTS% -Dlog4j.configuration=file:///PATH_URL/conf/startup_log4j.properties
set JAVA_OPTS=%JAVA_OPTS% -Duser.language=en

```

That would set the server language to English. Set the property to the appropriate locale value if you want a different language.

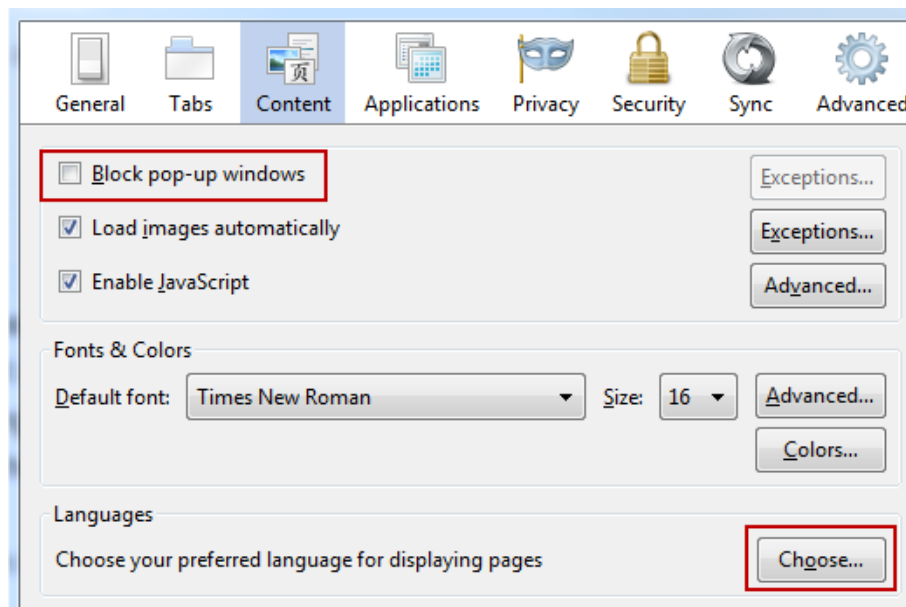
__2. **Save** your changes

2.1.4 Web UI Language

The Web UI uses the browser language to display information. Dependent on your install, that can also lead to mixed languages. You may want to configure your browser language preferences to match your expected language in RTC UI. Note that displaying a certain language in the Web UI relies on the specific language pack to be installed on the RTC server.

While this is a client configuration, you want to perform these steps at this point as this is the client you will use for the server post-installation setup process.

- __1. In Firefox open **Tools>Options**
 - __a. In the options navigate to **Content**
 - __b. If you have not yet done that, deactivate the *Pop-Up blocker*
 - __c. Click on the **Choose...** button in the *Languages* section at the end.



- __d. Add **English [en]** as language and use the move up button to bring it to the top.

2.2 Set Up the RTC Server

Set up the test server that will be used to experiment with process enactment.

- __1. Set up the CLM applications and JTS server
 - __a. Startup the JTS server by running
C:\JSWorkshops\IBM\JazzTeamServer\server\server.startup.
 - __b. Open a browser and run setup by navigating to <https://clm.process.ws/jts/setup>. Ignore any security warnings and add a security exception if asked.
 - __c. Log in to JTS using as login/password: ADMIN/ADMIN.
 - __d. Select *Express Setup*, then **Next**.
 - __i. In Configure Public URI, assure <https://clm.process.ws/jts> is the Public URI.
 - __a. Select *I understand that once the Public URI is set, it cannot be modified*, then select **Next**.
 - __ii. On the Create User page, create the PEW admin user then select **Next**:
 - __a. User ID: *pewadmin*
 - __b. Name: *pewadmin*
 - __c. Password: *pewadmin*
 - __d. Email: *pewadmin@bogus.ws*
 - __iii. Select **Next** when *Express Setup* is complete.
 - __e. On the Assign Licenses page:
 - __i. Under *Rational Team Concert*, next to *Rational Team Concert – Developer*, select **Activate Trial** if you are asked for activation.

You can optionally do the same for *Rational Requirements Composer – Analyst* and *Rational Quality Manager – Quality Professional*, if you deployed that applications as well.
 - __ii. Make sure the **Rational Team Concert – Developer** is checked, so it will be assigned to the *pewadmin* user.
 - __iii. Select **Finish**.
 - __f. On the Server Administration page, select the link for **Create Users** and enter the following values, then select **Save**:

- __i. Username: *Jim*,
- __ii. User ID: *jim*
- __iii. email address *jim@bogus.ws*
- __iv. Repository permissions: **JazzAdmins** (you can leave the default selected *JazzUsers* too).
- __v. Assign a License of type: **Rational Team Concert – Developer**.

You have set up your Jazz Team Server and the CCM application.

2.3 Set Up the RTC Eclipse Client Software

Process configuration can be done using the Web UI as well as the Eclipse client. Currently there are some features needed for process enactment not available in the Web UI. For this reason we need to configure an Eclipse client that will be used to do the Process customization.


2.3.1 Eclipse Client Language

The Eclipse Client picks up its language based on the language settings of the local machine. This can result in a mix of languages if you have installed language packs. Follow these steps if you want to enforce a language different than the one set on the machine.

__2. Modify the user setting:

- __a. Open the file **eclipse.ini** in <C:\JSWorkshops\IBM\TeamConcert> or if you installed the plain ZIP version in <C:\JSWorkshops\IBM\jazz\client\eclipse>

Set Other Jazz Clients Language



You can use the method described above to also set the SCM Command Line Interface, the .ini file is called **SCM.ini** and the Jazz Build System toolkit where the .ini file is called **JBE.ini**.

- __b. Add `-Duser.language=en` to the end of the file. It should now look like below

```
-Xms100m
-Xmx512m
-Dosgi.requiredJavaVersion=1.6
-Dosgi.bundlefile.limit=100
-Duser.language=en
```

Again, this sample setting would enforce the client language to English. Set the property to the appropriate locale value if you want a different language.

2.3.2 Eclipse Client User Preferences

Working with an Eclipse client has to be always performed in an Eclipse workspace context. You need to create a workspace and set the preferences for your work.

Eclipse Workspace Preferences



A basic set of RTC Eclipse preferences are instructed for the purpose of the workshop. You may want to configure a different set of preferences for your particular project needs.

- __1. Set up the Rational Team Concert Eclipse Client
- __2. If you did a Web Install or an Installation Manager Install, select **Start > All Programs > Rational Team Concert Client > Rational Team Concert Client**.

If you installed the plain ZIP version open the file **eclipse.exe** in *C:\JSWorkshops\IBM\jazz\client\eclipse*.

Starting the RTC Eclipse Client from the .exe file



You can also start the client by opening the eclipse.exe file if a Web or Installation Manager install method was used. Following the directions in Lab 1, the file will be located in *C:\JSWorkshops\IBM\TeamConcert*

- __a. When asked to select a workspace, enter *C:\JSWorkshops\Workspaces\Lab1*.
- __b. Close the *Welcome* view.
- __3. Set up preferences for RTC.
 - __a. Select **Window > Preferences**
 - __b. In **Team > Jazz Source Control > Check-in Policies**, select the following:
 - __i. *Auto check-in local changes*
 - __ii. *Perform check-in whenever a resource is modified*
 - __iii. Leave all other settings as-is.
 - __iv. Select **OK**.
 - __c. Configure external browser for Eclipse client:

External web browser in eclipse

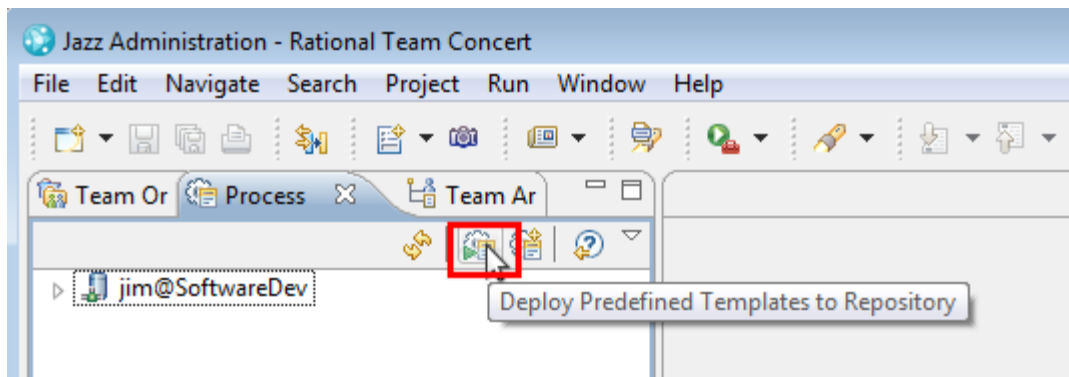
There is a bug in Rational Team Concert 4.0 where the client crashes when it tries to open the internal web browser. You modify this configuration in these steps to workaround the problem. See the following [link](#) for more information.

- __i. Open **Window > Preferences**
 - __ii. Navigate to **General > Web Browser**
 - __iii. Make sure you check *Use external web browser*. Select **OK**.
- __4. Create connections to the process and software development RTC repository
- __a. Switch to the *Jazz Administration* perspective.
 - __b. Select **Create a Repository Connection** in the *Team Organization* view. Enter the following information, and select **Finish**. Accept any certificates when prompted.
 - __i. URI: *https://clm.process.ws/ccm*
 - __ii. Name: *SoftwareDev*
 - __iii. UserID: *jim*
 - __iv. Password: *jim*
 - __c. Switch to the *Team Artifacts* view and you'll see the repository connections.

2.4 Create a Test Project

Process customization is always done against a project. This project is created in this section.

- ___1. Switch to the *Jazz Administration* perspective and select the *Process Template* view.
 - ___a. Deploy the process templates if necessary, by clicking the Deploy Predefined Templates icon. Select **OK** when prompted and wait for the operation to finish.



- ___b. Expand the repository connection node, right-click on the *OpenUP Process* template in the repository. Select **New > Project Area**.

Which process Template?

Alternatively you can use also one of the following process templates

- Formal Project Management Process
- Scrum
- Simple Team Process




If you do so, some screen shots and roles used in the following labs might not be available and you might have to modify the labs a bit.

Please note in RTC 5.0.1 the OpenUp and the Simple Team Process have been removed from the server and are available as separate download in the All downloads section of the product download pages.

- ___c. Create a project with the following attributes, then select **Next**:
 - ___i. Name: *Nifty Application Project*

- __ii. Summary: *The project for the Nifty application.*
- __d. Select *Automatically initialize the Project Area on Finish* as specified in the process template. Then select **Finish**.

Should you delay initialization?



Delaying process initialization is useful when you want to change the initialization information for the project. For example, if you want to add or remove work item templates that automatically create work items when the project is initialized.

- __2. The project area is created and a view of the project is displayed. Add the following customization:
 - __a. Expand the *Members* section of the *Overview* tab.
 - __i. Add the user *Jim* to the members and add the following process roles to *Jim* (in the order listed below, which will cause the *Project Manager* to be the first assigned role listed for *Jim*), then select **Finish**:
 - __a. Stakeholder
 - __b. Developer
 - __c. Project Manager
 - __ii. Make sure the roles are top-down Manager, Developer, Stakeholder and everyone.
 - __b. **Save** the project.
 - __c. Review the project and its process.
 - __i. Categories
 - __ii. Timelines
 - __iii. Process Configuration

2.5 Summary

Congratulations! You've set up your process development environment to enact your process by setting up the RTC server and client environments, and creating a project. You now have the following in your process development environment:

- A process development environment with a JTS and associated CCM application.
- A project, Nifty Application Project that you can manage the development of your process.

2.6 Appendix A – Additional Considerations

Please find some additional topics you should consider when setting up your servers.

- **Other Tomcat TCP Ports**

Tomcat uses other ports as shutdown port etc. If you want to run several Tomcat servers on one machine you have to make sure that the instances use different ports, including as shutdown ports.

- **Tomcat Shutdown Password**

You want to change the default shutdown password, otherwise anyone with a Tomcat install can shutdown your server using the default password and port.

- **VMWare 8**

Uses 443 for VM sharing. Disable sharing in VMWare or select another port.

Lab 3 Configuring Work Items

This lab will guide you through the work items configuration process by the definition of a new Work Item Type within a new custom Work Item Type Category in Rational Team Concert. You will define the basic structure and information for this new type.

Lab Scenario

Acme Corporation wants to embrace new technologies in their applications evolution and to be more responsive to customer needs. The team working in “Nifty Application 1.0” has been discussing how the application should evolve and how the architecture will adopt these new technologies. In addition, new feature requests have been received that will require extending the application's technology. The team has determined that they need to dedicate some development effort to exploring experimental changes in the application. The team needs to define how this effort should be tracked as part of their work process.

In this lab you will:

- Review and understand the new process needs that you have to implement in RTC
- Create new work item types and type categories, understanding the difference between them
- Create new attribute types and reuse existing attributes
- Create and understand work item's workflows, and how to link workflows with work items
- Create and adjust the presentation of your work items

Lessons Learned:

- Create work item types and customize all the basic elements involved in their definition

3.1 Customize the Process in RTC

As the Project Leader of Acme in the context of Nifty Application Project, you have been assigned the task of modifying the process of your project. The project requires a new work item type “Technology Review” with a dedicated workflow and several custom attributes. You will also set the permission for roles to update information at the work item.

3.1.1 Understand the new needs

Your process engineers have provided you with some information about the required process changes. Acme uses Rational Method Composer to describe the organization's processes, so your process engineers have provided you with the new process description elements for you to implement in RTC. Review this information and get back to it, in case you need to.

The information below describes the Technology Review work product, that you need to implement in your RTC process. The information also documents which role is responsible for modifying this work product; the developer.

Work Products > Development > Technology Review

Artifact: Technology Review

An exploration of new technology to understand the implications of adoption.
Domains: [Development](#)

Purpose

- To understand the implications of adopting a new technology.
- To support a decision on whether or not the new technology should be adopted.

[Back to top](#)

Relationships

Roles	Responsible:	Modified By:
		<ul style="list-style-type: none"> Developer
Tasks	Input To:	Output From:
		<ul style="list-style-type: none"> Explore Technology

[Back to top](#)

A formal description of this work product allows you to understand its lifecycle and how it will fit in the overall team work needs.

[-] Description

Main Description

This artifact manages the adoption (or rejection) of a new technology.

Exploring a new technology usually begins with a proposition based on an identified area of improvement in the product or a brand new technology that gains relevance in the industry. Begin researching to familiarize yourself with the technology.

Experiment to test the technology, and if those experiments indicate the technology will work then it can be approved for use in the product. Finally, adopt the technology by incorporating it into the product and testing to assure it fulfills its purpose.

Any new technology to incorporate may be rejected based on the information gathered:

it's not accurate to what the product needs; or based on the tests performing during the experimentation, if the technology demonstrates too much impact on the actual state of the product.

Along with its formal description, Key Considerations for the Work Product are documented. Reading carefully these considerations you can deduce the basic set of attributes that you need to implement:

[-] Key Considerations

Important aspects when reviewing the adoption of the new technology are:

- The complexity of its adoption. This complexity and the state of the development will determine the degree of impact that the adoption will have.
- The different costs that the adoption of the technology would have in the project
- Which departments would be impacted during this adoption work, that a solution adoption planning will have to consider.

[⤴ Back to top](#)

[-] Tailoring

Impact of not having

New technologies may not be fully understood and not adopted when they could provide value.

Technologies that are ineffective or inappropriate may be adopted into the project.

Reasons for not needing

Technologies and platforms that are well understood do not need technology reviews.

[⤴ Back to top](#)

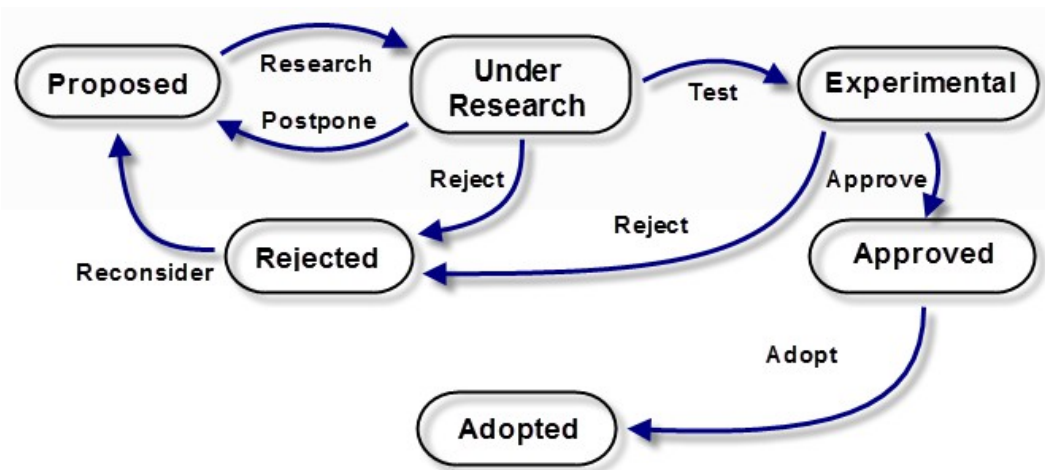
Your process engineers have provided you with the Work Product workflow, based on its formal description. It represent its lifecycle that you will have to implement in RTC:

Workflow Table

WorkProduct Workflow Table	Adopted	Approved	Experimental	Proposed	Rejected	Under Research
Adopted						
Approved	Adopt					
Experimental		Approve			Reject	
Proposed						Research
Rejected				Reconsider		
Under Research			Test	Postpone	Reject	

[Back to top](#)

Here is an identical description of the workflow that you need to implement. It is presented in a different way that makes easier to understand the different transitions:



3.1.2 Create a new Work Item Type with Custom Attributes

In this section you will create the new work item type Technology Review and add custom attributes you need to implement your business logic, based on the description provided in the previous section.

- __1. Create a Work Item Type. You are going to model the work product Technology Review as a new type of work item.
 - __a. Open the Rational Team Concert eclipse client if not already open, and switch to the *Work Items* perspective.
 - __b. Open the project editor if needed by right-clicking *Nifty Application Project* and selecting **Open** in the *Team Artifacts* view.
 - __c. Open the *Process Configuration* tab, and navigate to **Project Configuration > Configuration Data > Work Items > Types and Attributes**.
 - __d. Click **Add...** in the *Types and Attributes* section in the right side. Enter the following values and select **OK**:
 - __i. Name: Technology Review
 - __ii. ID: com.acme.openup.workitem.workitemType.technology_review
 - __iii. New Category (CHECKED):
com.acme.openup.workitem.workitemTypeCat.technology_review

What is the deal with the Ids?

When customizing your process you will have to define IDs for different items: work item types, attribute types, presentations, ...



It is a good practice to define a naming convention within your enterprise. The elements of process customizations should follow this naming convention so the RTC process is consistent in your company, and it will ease the development leveraging these IDs: you will be able to locate process customizations within the whole process. In addition, as removing elements is an operation to avoid for its potential impact, it is also a good practice to instead modify the ID of the elements you want to deprecate with a certain pattern.

What is a Work Item Type Category?



Work Item Type Categories define a common workflow and custom attributes to be shared by all the work item types that belong to it.

As our new work item type will have its own workflow, we need to create a new category along with the type.

- __e. Click the *Icon* drop-down and select **architectural.gif**
 - __f. **Save** your changes.
- __2. Add Custom Attributes. You have reviewed the Key Considerations section of Artifact: Technology Review (3.1.1 Understand the new needs on page 36) and have identified the custom attributes to define in Rational Team Concert for the new work item type. Perform the following steps to create them:
- __a. Select the **Types and Attributes** node in the *Configuration Data > Work Items* tree if not already there
 - __b. Make sure Technology Review Work Item Type is selected
 - __c. Navigate to the **Attributes** section at the bottom of the page.
 - __d. Select **Add ...**
 - __e. Enter the following values:
 - __i. Name: Technology Name
 - __ii. ID: com.acme.openup.workitem.attribute.techname
 - __iii. Type: Medium String
 - __iv. Leave the rest of the parameters with the default value
 - __v. Select **OK**.
 - __f. Repeat steps d-e with the following information:

Name	ID	Type
Estimated Effort	com.acme.openup.workitem.attribute.esteffort	Small String
Cost	com.acme.openup.workitem.attribute.cost	Small String

Estimated Investment	com.acme.openup.workitem.attribute.estinvest	Small String
Total Cost	com.acme.openup.workitem.attribute.totalcost	Small String
Workflow Information	com.acme.openup.workitem.attribute.wkinfo	Wiki
Affected Departments	com.acme.openup.workitem.attribute.affdepts	String List
Responsible Architect	com.acme.openup.workitem.attribute.resparch	Contributor

String type for cost attributes?



You may wonder why you are defining these attributes' types as String instead of Integer. It's all for the sake of customization exercises you will perform in the following labs.

__3. Reuse existing attribute for Impact information:

- __a. Select **Add ...**
- __b. Check **Reuse Existing Attribute** and select "Impact – type: risk (Enumeration)" from the drop-down list
- __c. Select **OK**.
- __d. **Save** your changes.

Re-using attributes



This reuse feature allows you to leverage attribute customizations from other existing work item type categories, easing maintenance of information.

__4. Define enumeration for the complexity attribute:

- __a. Select the **Enumerations** node in the *Configuration Data > Work Items* tree
- __b. Select the **Add...** button located below *Choose the Enumeration to edit*

- __c. Enter the following values and select **OK**:
 - __i. Name: Complexity
 - __ii. com.acme.openup.enum.complexity
 - __iii. Process Specification: CHECKED

- __d. Select **Add...** at the *Enumeration Literals* section, and enter the following values for the “Name” attribute of each literal value, you can leave the “External Value” field empty: (assign icons as you will)
 - Low
 - Manageable
 - Complex
 - Unapproachable
 - __i. Select the value **Manageable** for *Default Literal* and *Unassigned Literal*










- __e. Create the attributes for the enumeration:
 - __i. Navigate back to the **Types and Attributes** node in the *Configuration Data > Work Items* tree
 - __ii. Make sure *Technology Review Work Item Type* is **High-lighted**
 - __iii. Navigate to the *Attributes* section at the bottom of the page.
 - __iv. Select **Add ...**
 - __v. Enter the following values, then select **OK**:
 - __a. Name: Complexity
 - __b. com.acme.openup.workitem.attribute.complexity
 - __c. Type: Complexity (Enumeration)

- __f. **Save** your changes.

- ___g. At the top of the *Attributes* section, if you check **Show only custom attributes**, you should see something like the following:

▼ **Attributes**

Show only custom attributes

Name	Type
 Affected Departments	String list
 Complexity	Complexity (Enumeration)
 Cost	Small String
 Estimated Effort	Small String
 Estimated Investment	Small String
 Impact	Risk (Enumeration)
 Technology Name	Medium String
 Total Cost	Small String
 Workflow Information	Wiki

Adding attributes to existing work items



If you add new attributes to work item types that already have work items created, you have to Synchronize the attributes to be able to use the new attribute in that existing work items.

The article *Cool "Hidden" Features in Rational Team Concert: Part 2* explains this feature. [Part 3](#) contains links to the other articles (attributes synchronization is located in [part 2](#) of the series)

3.1.3 Create a Workflow

In this section you will create the workflow for the Technology Review work item type you just created, and connect it to the work item type.

- __1. Define the workflow. Your organization's Process Engineer described the transition table matrix that will govern the lifecycle of the work product. Now you will create a work item workflow matching that description.
 - __a. Select the **Workflows** node in the *Configuration Data > Work Items* tree.
 - __b. Select **Add...** next to the *Choose the workflow to edit* field.
 - __c. Enter the following values and then select **OK**:
 - __i. Name: Technology Review Workflow
 - __ii. ID: tecReviewWorkflow
 - __d. Navigate down to the States section. Click **Add...** button and create the states with the following information: (assign icons as you will)

Name	Group	Show Resolution
Proposed	Open	Unchecked
Under Research	In Progress	Unchecked
Experimental	In Progress	Unchecked
Approved	In Progress	Unchecked
Rejected	Closed	Checked
Adopted	Closed	Unchecked

What are the State Groups?

The State Groups help you categorize states and resolutions that relate to each other.



You can create new state groups and associate them to the default group categories: Open, In-Progress, Closed. These grouping of states and resolutions are very useful for building queries / reports, and when you extend Rational Team Concert and want to interact with Work Items states.

In addition, these categories are associated with OSLC state groups, thus providing a way of isolating the detailed knowledge of your actual defined workflow for integration with other systems via OSLC.

- __e. Create the actions in the transition matrix: navigate to the Transitions section of the page and perform the following changes:
- __i. In the combination From “Proposed” To “Under Research”, click the combo box and select **New Action...**
 - __ii. Enter *Research* as the name and select **OK**.
 - __iii. Repeat steps i – ii, with the rest of the actions until you build the following transition table (which maps with the table matrix described in Rational Method Composer provided to you earlier in this lab in 3.1.1 Understand the new needs on page 36):

▼ Transitions		Proposed	Under Research	Experimental	Approved	Rejected	Adopted
From	To						
Proposed		<None>	Research	None	None	None	None
Under Research		Postpone	<None>	Test	None	Reject	None
Experimental		None	None	<None>	Approve	Reject	None
Approved		None	None	None	<None>	None	Adopt
Rejected		Reconsider	None	None	None	<None>	None
Adopted		None	None	None	None	None	<None>

- __f. Define the start action:
- __i. Select the combo box next to *Start Action* and click **New Action...**
 - __ii. Enter *Propose* as name and *Proposed* as Target State. Select **OK**.

Resolve and Reopen actions



You may wonder why we left these fields blank. Our defined workflow has two different possible end states depending on the path: we didn't consider a unique and global resolve action.

In addition, we defined a reopen type of action ("Reconsider") for just one of the closed states.

- __g. Define the resolutions: you want to further explain the possible rejections of a new technology.
 - __i. Navigate to the Resolutions section.
 - __ii. Click **Add...**
 - __iii. Enter *Inaccurate* as the name and *Closed* as the Group. Select **OK**.
 - __iv. Repeat the steps to include a resolution called *Extreme Impact* member of Closed Group.
- __h. Navigate back to the *Actions* section of the page and configure resolutions:
 - __i. Select the *Reject* action in the left panel
 - __ii. Mark the check box next to both resolutions in the right side.

Actions

Configure the Actions for this Workflow. For each Action that will have Resolutions, select those Resolutions and the order in which they will appear.

Name	Target State	Description	Resolution	Group
Adopt	Adopted		<input type="checkbox"/> Inaccurate	
Approve	Approved		<input checked="" type="checkbox"/> Extreme Impact	
Postpone	Proposed			
Propose	Proposed			
Reconsider	Proposed			
Reject	Rejected			
Research	Under Research			
Test	Experimental			

Buttons: Add..., Edit..., Remove

- __i. **Save** your changes.
- __j. Associate the workflow to the work item:
 - __i. Navigate to the **Types and Attributes** node under *Work Items*
 - __ii. Choose *Technology Review* under *Work Item Types*

- ___iii. Navigate to *Workflow* section, expand the combo and select **Technology Review Workflow**
- ___iv. **Save** your changes.

3.1.4 Customize the Presentation For the Work Item Type

To be able to use the custom attributes at the Technology Review work item type, you will now configure the editor presentations for the work item type.

- __1. Customize the presentation for the work item.
 - __a. Select the **Editor Presentations** node in the *Configuration Data > Work Items* tree.

Work Item editor presentation structure:

The work item editor presentation is structured in **tabs**. Each tab contains **slots** which are the containers for the actual presentation of information. This information can be “attribute-based” or “non-attribute” based (like the presentation for the attachments of a work item).



How information is placed in a tab, and the organization of the slots, is based on the concept of a **layout**. There are different types of layouts depending the information you want to place and how you want to organize it. For more information see [Work item editor presentations](#).

The presentation editor is based on the concept of reuse. The definition of tabs and slots can be shared among different work item editor presentations. If you modify a shared configuration it will apply to all the presentations that make use of it. As an alternative, you have the option of creating new presentations (whether tabs or layouts), or duplicating an existing presentation and customize for your particular needs.

- __b. For ease of use, you will create a presentation based on the existing one, customizing it to hold the required attributes. Make sure in the *Choose editor presentation to edit* box the `com.ibm.team.workitem.editor.default` is selected, and select **Duplicate ...**
- __c. Enter the following ID and select **OK**: `com.acme.openup.workitem.editor.techreview`
- __d. Adjust presentation of attributes: you based your presentation in the default one which is used by Task work items. As the Technology Review work item is a specialized type of task, you will be just adding a specialized tab for the new information and reusing the rest of the editor presentation.
 - __i. Select **Add Tab ...** and enter the following values, then select **OK**.
 - __a. Title: Tech Review Details
 - __b. Layout: Custom Attributes Layout

- __c. Create Tab ID: CHECKED
- __d. ID: com.acme.openup.workitem.tab.techrevdetails
- __ii. High-light the new created tab and select **Add Section** Enter the following values, then select **OK**.
 - __a. Title: Adoption Cost Estimations
 - __b. Slot: Left
 - __c. Create Section ID: CHECKED
 - __d. ID: com.acme.openup.workitem.section.techcostdetails
- __iii. Repeat the previous steps to add a new section with the following details:
 - __a. Title: Tech Review Miscellaneous
 - __b. Slot: Right
 - __c. Create Section ID: CHECKED
 - __d. ID: com.acme.openup.workitem.section.techgeneralinfo
- __iv. Highlight *Adoption Cost Estimations* section and click **Add Presentation...** for each of the items listed below. (do it in the order of appearance. All of them are "Attribute based"):

Attribute	Kind	Label	Description
Affected Departments	String List	NA	NA
Estimated Effort	String	NA	Measured in developers per day
Cost	String	Average Cost	Person hour cost
Estimated Investment	String	NA	Material resources needed estimation
Total Cost	String	NA	NA

- __v. Now highlight **Tech Review Miscellaneous** section and click **Add Presentation...** for each of the items listed below (do it in the order of appearance):

Attribute	Kind	Label	Description
Technology Name	String	NA	Descriptive name of technology under study
Workflow Information	Wiki	NA	NA
Complexity	Enumeration	NA	NA
Impact	Enumeration	NA	NA
Responsible Architect	Contributor		

- __e. Highlight the *Tech Review Details* tab and click **Move Up** until you place it just after the *Overview* tab.
- __f. **Save** your changes.
- __g. Link the presentation to the work item:
 - __i. Navigate to the **Types and Attributes** node under *Work Items*.
 - __ii. Make sure *Technology Review Work Item Type* is **High-lighted** under *Work Item Types*
 - __iii. In the *Work Item Editor* box, select `com.acme.openup.workitem.editor.techreview`
 - __iv. **Save** your changes.

Different types of editors?

You may noticed that there are other type of editors in this page. The main differences are:



- Work Item Editor: the main editor for managing work items. Used in the different clients when you open a work item.
- Inline Work Item Editor: Inline editors are presented in the query results in Web UI
- Lightweight Work Item Creation Dialog: presentation for wizards that allow a quick creation of work items, like when delivering change sets or from other integrated CLM applications.
- Plan Editor Preview: presentation for modifying work items from within a plan.

For all but the main editor, the default one usually is enough for the needs of your project, as they present the minimum basic set of attributes. However, you can also customize the other editors to your needs as we did in the example for the full editor presentation.

__h. Check the presentation.

__i. Select the arrow next to the New Work Item icon:



- __ii. Select **Nifty Application Project > Technology Review**. A new work item appears
- __iii. Select the **Tech Review Details** tab. It should look like the following:

The screenshot displays the 'Technology Review <01:30:32>' interface. At the top, there is a title bar with a 'Save' button and several utility icons. Below the title bar is a 'Summary:' section with a text input field, a dropdown menu set to 'Uninitialized', and another dropdown menu. The main content area is divided into two panels: 'Adoption Cost Estimations' and 'Tech Review Miscellaneous'. The 'Adoption Cost Estimations' panel includes a large text area for 'Affected Departments', 'Estimated Effort:', 'Average Cost:', 'Estimated Investment:', and 'Total Cost:' fields, along with 'New...' and 'Remove' buttons. The 'Tech Review Miscellaneous' panel includes a 'Technology Name:' field, 'Workflow Information:' text, and dropdown menus for 'Complexity:' (set to 'Manageable') and 'Impact:' (set to 'Medium'). At the bottom, a navigation bar contains tabs for 'Overview', 'Tech Review Details', 'Links', 'Approvals', and 'History'.

3.1.5 Configure Permissions

To make sure only certain roles are able to change the new custom attributes created for the *Technology Review* work item type you will now set the role based permissions. You will also specify the roles that will be able to execute the transition actions of the workflow you just created. The main editor for this artifact is the developer (see 3.1.1 Understand the new needs on page 36).

- __1. Adjust permissions for attributes and workflow states: you have to give permissions to the roles for the new items created in the process.
 - __a. Return to the project area editor and select **Permissions** under *Team Configuration* node.
 - __b. Check the option to **Show all actions and roles**
 - __c. According to the information you were given (see 3.1.1 Understand the new needs on page 36), the Developer is the role responsible of modifying the *Technology Review* work product.
In addition, you will grant the Tester role the needed permissions for the phases of lifecycle in which his collaboration is required for the *Technology Review*. To adjust the process:
 - __i. Adjust permissions to the attributes:
 - __a. Open the node *Work Items > Save Work Item (server) > Modify the work item*
 - __b. Give the Developer role the only one to modify the created attributes (remove the permissions to the Project Manager role):

	Everyo...	Analyst	Architect	Developer	Project Manager	Stakeh...	Tester
... Modify the work item's 'Filed Against' attribute	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
... Modify the 'Affected Departments' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Complexity' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Cost' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Customer Concern' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Estimated Effort' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Estimated Investment' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Responsible Architect' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Technology Name' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Total Cost' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Workflow Information' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Rank (relative to Priority)' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
... Modify the 'Maximal Estimate' attribute	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

- __ii. Adjust permissions for workflow:
 - __a. Open the node *Work Items > Save Work Item (server) > Create a work item > Create a work item of a specific type*

- ___b. Under Create a 'Technology Review' work item, **CHECK** the boxes for the Developer and Tester roles, and **UNCHECK** for the rest of the roles.
- ___c. Open the node *Save Work Item (server) > Trigger a workflow action*
- ___d. Make sure the Developer role is the only one with permission to modify the actions which have in parenthesis the string "(Technology Review workflow)". Give also permission to the Tester role for the actions of *Propose* and *Reject*.

Actions	Everyone (d...	Analyst	Architect	Developer	Project Man...	Stakeholder	Tester
Research (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Postpone (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Test (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Reject (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Approve (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Adopt (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Reconsider (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Propose (Technology Review Workflow)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

___d. **Save** your changes.

3.2 Gather the process changes

After successfully customizing the process in a living project, you want to gather this changes back into the templates managed in your process development environment, so new projects can be created with these new items already defined.

Customizing and testing in my production project?

The configuration steps in this lab were directly performed in the Nifty Application production project by the project leader, to focus in the features introduced. In your real production environment you, as project leader, would have a testing environment in which you validate process changes before applying them in production. This test environment, depending on your needs, could range from just a test project area in the production server, or a complete test installation.

Replicating process customization changes from your test environment in you real production project area can be performed in any of these ways:



- The Project Lead replicates all the modifications from the test project area. For this operation, the enterprise wide naming convention for the IDs of process elements outlined as best practice previously in this lab will be very useful.
- If test and production project areas are always aligned, Project Lead can copy the entire process source XML from test and replace the one in production project area. This operation is to be performed with extremely care.

We have omitted this test project area intermediate step for the sake of workshop timing and to concentrate on the main operations of the lifecycle.

- __1. Generate the process template from the Project Area:
 - __a. Switch to the *Jazz Admin* perspective, and open the *Team Artifacts* view
 - __b. Right-click the project area *Nifty Application Project* and select **Extract Process Template...**
 - __c. Give it the following values and select **Finish**.
 - __i. Name: Acme OpenUp Nifty Based
 - __ii. ID: openup.process.acme.nifty.ws

__iii. Summary: Acme Corporation's OpenUP Process

What if you want to reuse a template ID?

You cannot create a process template from a Project Area and assign it an ID of a process that already exist in the repository. On the other hand, you can't delete a process template from the repository if a project area exist that was created based on it.



If you want to keep one corporate process template (and thus just one ID), you can generate a process template from a project area with the process shown here, export it and then re-import it giving it the existing ID; you will then be prompted to overwrite.

Important! A process template generated from a project area doesn't consider localization, but is generated in the language of the project area instantiation.

__2. Export the Process Template:

- __a. Back in the Process Templates view, right-click your newly created template called *Acme OpenUp Nifty Based* and select **Export**.
- __b. Select *C:\JSWorkshops\Downloads* as the directory and click **Finish**.

Export and Import of Process Templates

You can export the process into a folder or you can export it as archive file.

Exporting as a folder is useful if using an SCM system to version the process template.



Exporting as an archive file is easier for sharing the process by sending it e.g. via e-mail.

You can use the RTC Eclipse Client to import process templates from Folder or Archive File Exports.

You can use the Templates menu in the Web Administration UI to import process template. The Web UI only supports to import Archive Files.

Importing a Process Template



If you want to import a process template into a repository that has a template with the same ID already, you have two options, you can overwrite the existing template or provide the newly imported process template a new ID.

Since you can't delete process templates that are referenced by project areas, the cleanest solution is to overwrite the existing process template. This helps maintaining only a short list of process templates to be used and avoids using the old version.

Process Templates and Project Area Process



When a project area is instantiated, the chosen process template is copied into the project area. Changing the template has no effect on the existing project areas using the template.

If you need to maintain a common process and want to be able to align existing project areas with it, use [Process Sharing](#).

Process sharing allows a project area created with the “Unconfigured Process” template to use the process definition from another project area in which is maintained. You can change the process of that process area and all the ones that share the process will pick up the change.

You can create a new project area with a modified process and change the process area you share the process from to inherit the modification. Currently the capability of modifying the process in areas that share a process very limited. Usually customizing a domain (e.g. WorkItems) means copying the process and loosing the sharing capability for the domain that was overwritten. You can check [Managing fine grain process customization for work items in RTC 4.0.1](#)

3.3 Summary

In this lab you have successfully created a new work item type Technology Review. You have

- Created the work item type
- Added custom attributes
- Created a new workflow for the work item type
- Modified the editor presentations to make the custom attributes available for editing
- Adjusted the permissions for the attributes of the Technology Review work item type
- Created a new process template to make your changes available enterprise wide

Lab 4 Work Item Customization

This lab will step you through some advanced work item attributes customization provided by Rational Team Concert.

Rational Team Concert work items provide a rich set of attribute types and values for different needs. These needs may change as your process matures. For example, it is possible to:

- Create your own types for custom values such as enumerations
- Provide default values based on context such as the value of other attributes
- Reduce the number of items to select in an attribute based on context
- Populate possible values by pulling data from an external system
- Validate attributes

Lab Scenario

You are helping your team customize a Rational Team Concert process. Your team requires



- Default values for attributes
- Calculated values and value sets for attributes
- Validators for attributes that make sure the format of the data is correct when saving a work item.

It's your job to do all customization for the project team.

Suggested Reading

For more information please read



- [Getting Started with Work Items in Rational Team Concert](#)
- [Customizing attributes in Rational Team Concert 4.0](#)
- [Customizing attributes in Rational Team Concert 3.0](#)
- [Wiki entry: Work Items attribute customization](#)

Supported UI's



For convenience the lab will use the Eclipse Client to develop and test the customization. However the customization works with the Eclipse as well as with the Web UI.

Java Script Based Customization




Please note, that Java Script Based customization will be introduced in a separate Lab.

4.1 Default Value Provider

Sometimes it is desirable to set reasonable defaults for some of the other work item attributes. It is often possible to set default values for some enumeration type work item attributes in the process configuration. However, there are several attribute types where this is not possible.

Rational Team Concert starting with 3.x provides some basic capabilities to set default attribute values. The following sections describes default value providers that can be created with RTC 4.0.

4.1.1 Default values for attributes of type Contributor



Section Scenario

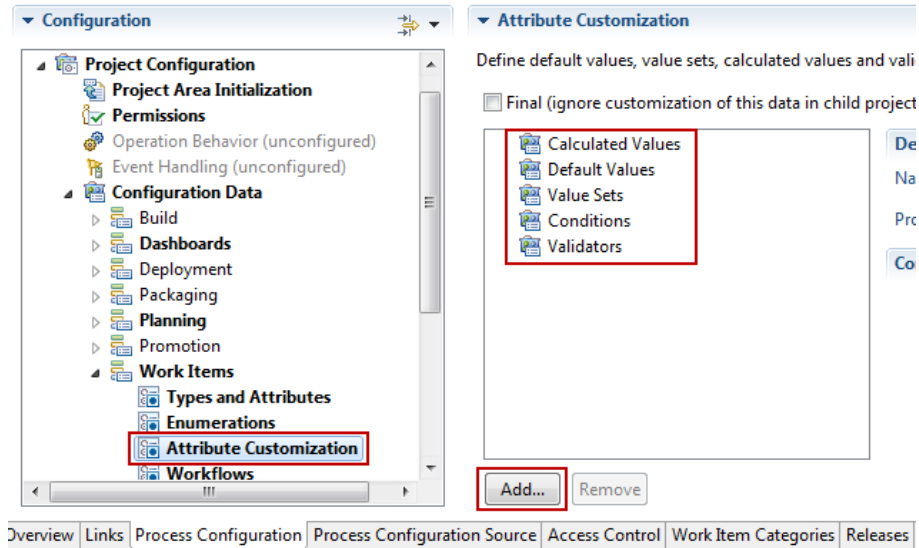
RTC work items provide attributes for user entries such as the Owner, who is responsible for the work item. It is possible to add additional attributes of type Contributor that contain user information to help implement business requirements. One common request is to be able to set a default for these kind of attributes.

Your project has decided that each work item you create must be initially owned by the creator of the work item. You have been given the task to implement this.

- __1. Open the RTC Eclipse Client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1\`. Log in as user **jim** password **jim**.
- __2. Go to the *Team Artifacts* view.
- __3. Right click at the **Nifty Application Project** project area in the *Team Artifacts* view and select **Open** to open the *Project Area Editor* for your project.
- __4. Switch to the *Process Configuration* tab and navigate to the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.

__5. Create a new Default Value Attribute Customization Configuration.

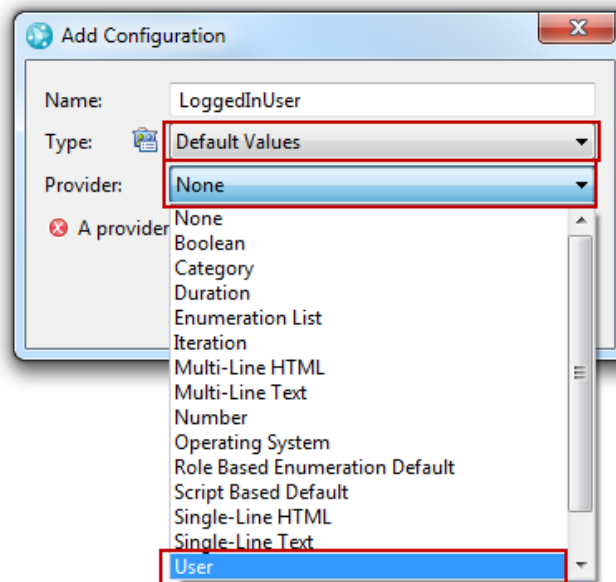
__a. In the *Attribute Customization* editor, click the **Add** button at the bottom of the editor to add a new Attribute Customization configuration.



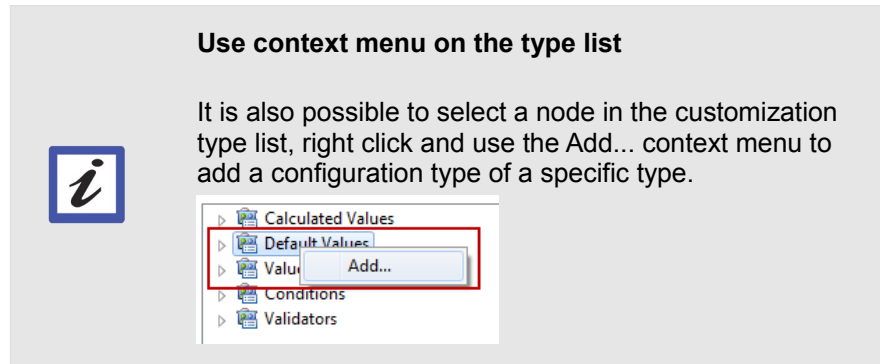
__i. Enter **LoggedInUser** as *Name* of the configuration.

__ii. As *Type* select **Default Values**.

__iii. Use the *Provider* selection drop down box to select a provider. Read through the list of available providers and select the provider **User**. The drop down list shows the available built-in provider for the selected configuration *Type*. For *Default Values* RTC has configurable built-in providers for several attribute types.



__iv. Click **OK**.



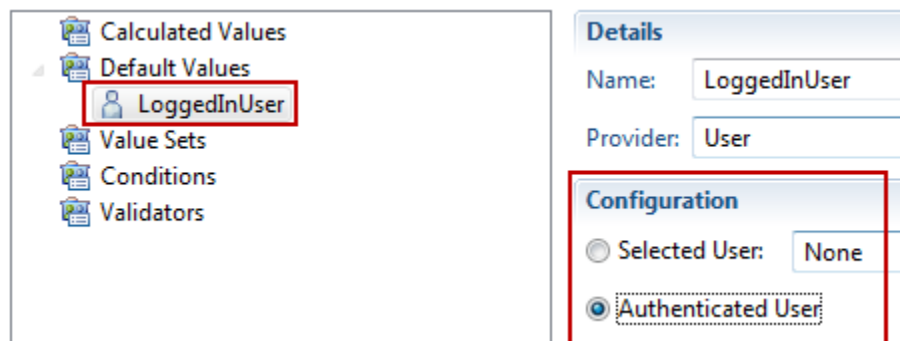
- __b. Now the new default value attribute Customization is created. To make it work it needs additional configuration which is done in the Configuration editor to the right.

If the configuration editor is not visible, the Customization is not selected. Expand the Default Values node if necessary to see the *LoggedInUser* Attribute Customization you just created to see the configuration editor.

- __i. Review the current configuration. There are two options:

- Select a specific user
- Select the **Authenticated User**. The Authenticated User is the user that is currently logged in

- __ii. Select the Authenticated User radio button.

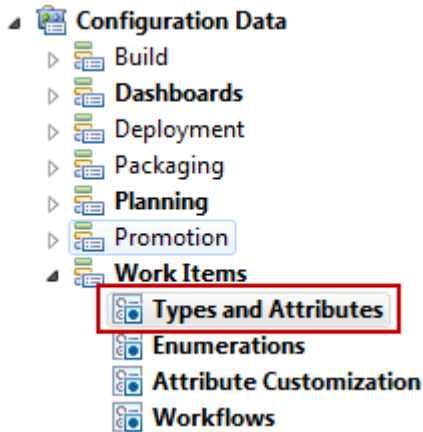


- __c. **Save** the change to the process configuration.

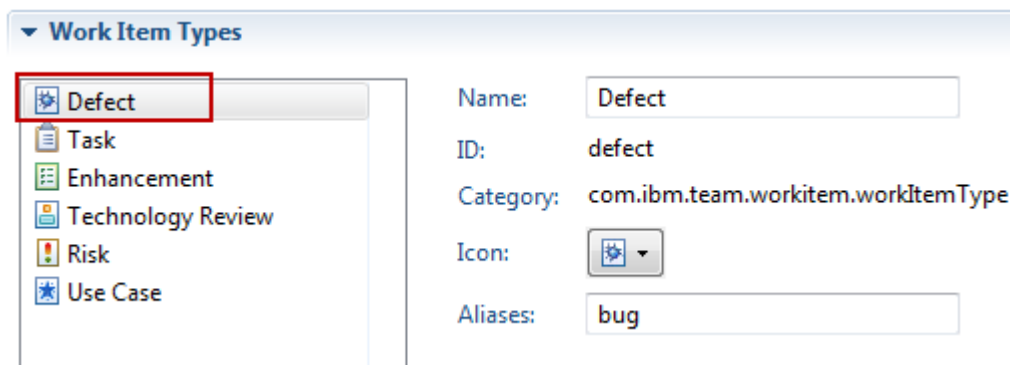
You have successfully created a Default Value Provider.

- __6. Now the value provider needs to be configured for the attribute:

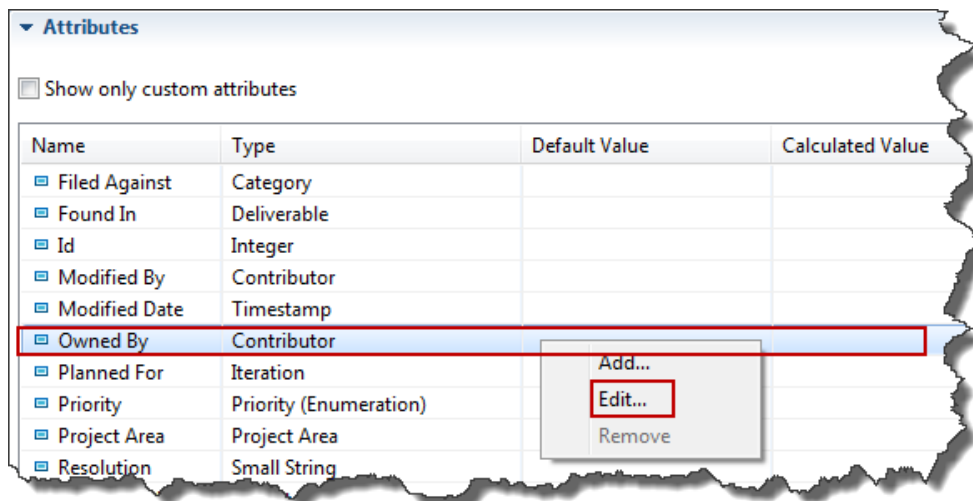
- ___a. In the Project Area editor Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.



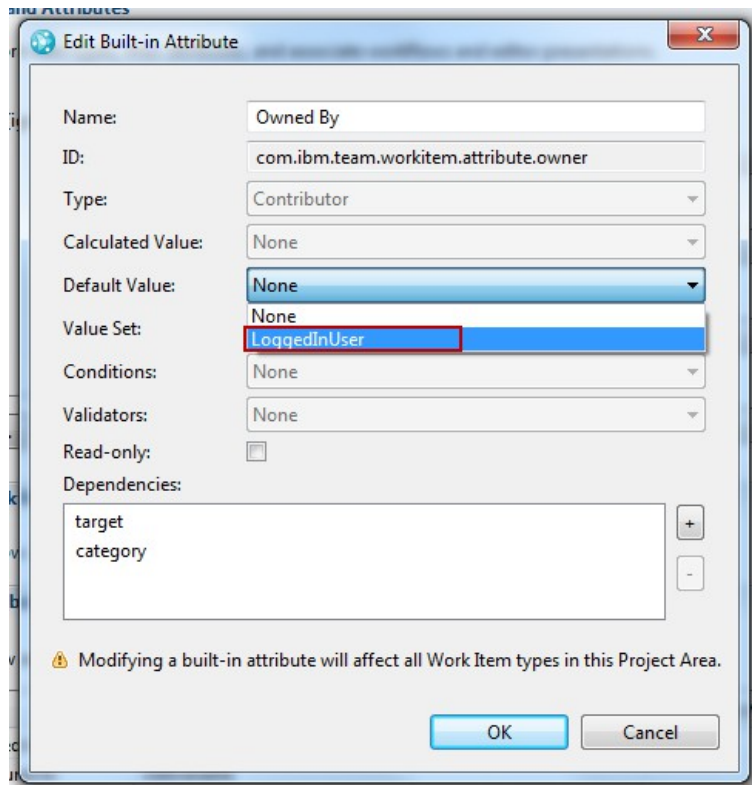
- ___b. Select any work item type that has the attribute you want to configure. In our case for example the work item type *Defect*



- ___c. Scroll down to the Attributes section, browse for the Attribute named *Owned By*. Right-click the row and select **Edit...**



- ___d. The attribute editor comes up and displays the properties of the attribute. Note that the *Default Value* section is now enabled. Click at the drop down list and select **LoggedInUser**.



Buttons are only active if a provider is available



The drop down lists for Calculated Value, Default Value, Value Sets, and Validators become available for selection as soon as a matching value provider is available.

Attribute Customization affects attributes across all work item types



Currently Attribute Customization affects all work items that use the attribute that is customized. That means you can not have different behaviors for the same attribute used in different work item types.

This limitation can be avoided in some cases using Java Script based Attribute Customization introduced in the next lab.

- ___e. Make sure that the default value provider **LoggedInUser** is selected and press **OK**.

__f. The columns should now show the new default value provider.

Modified By	Contributor	
Modified Date	Timestamp	
Owned By	Contributor	LoggedInUser
Planned For	Iteration	
Priority	Priority (Enumeration)	

__g. **Save** your changes to the process configuration.

__7. Test the new Default Value Provider.

__a. Create a work item such as a Defect, Task, Story or Epic.

__b. Check that the Owned By attribute now defaults to the current user.

The screenshot shows a 'Details' panel for a work item. The 'Owned By' field is highlighted with a red box and contains the name 'Jim'. Other fields include Type (Defect), Severity (Normal), Found In (Unassigned), Team Area (Nifty Application Project Team), Filed Against (Unassigned), Tags (empty), Priority (Unassigned), and Planned For (Unassigned).

You have successfully created the default value for the owner of the work items.

4.1.2 Role Based Enumeration Default

Section Scenario

Your project would like to make sure that the default priority of a work item is based on the role of the user.



- Work items created by the Project Manager and Architects should default to “high” priority.
- Work items created by an Analyst or Developer should default to “medium”.
- Other project roles should default to “low”
- For the default role, the priority should be “unassigned”.

You have been identified as the resource that can help the project to fulfill this requirement. Though puzzled over why projects would come up with requirements like this, you start to implement it right away.

- __1. Open the Eclipse Client if it is not already opened and connect to your repository. Log in with the user **jim** password **jim**. Open the Project Area editor for your Project.
- __2. Switch to the Project Area editor Process Configuration tab. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __a. Press the **Add** button at the bottom of the editor to add a new configuration.
 - __i. As Name enter *Default Priority by Role*.
 - __ii. As Type select *Default Values*.
 - __iii. As Provider select *Role Based Enumeration Default* from the drop down list.
 - __iv. Select **OK**
 - __b. The Editor for the Attribute Customization configuration should now show the *Details* and the *Configuration* editor section editor to the right.

- __i. Use the **Select...** button and select the enumeration named *priority*.

The screenshot shows a configuration editor with two main sections: 'Details' and 'Configuration'.
In the 'Details' section, there are two input fields: 'Name' with the value 'Default Priority By Role' and 'Provider' with the value 'Role Based Enumeration Default'.
In the 'Configuration' section, there is an 'Enumeration' field with the value 'priority' and a 'Select...' button to its right. The 'Select...' button is highlighted with a red rectangular box.

- __c. The configuration editor now shows the roles available in the order defined in the process configuration.

Select the default priority Literal for each role.

- __i. Project Manager and Architect: High priority
- __ii. Analyst and Developer: Medium priority
- __iii. Stakeholder and Tester: Low priority
- __iv. Default and other roles: Unassigned.

- __d. The configuration should now look as follows. You might have additional roles that are not presented in the screen shot below.

Details

Name:

Provider:

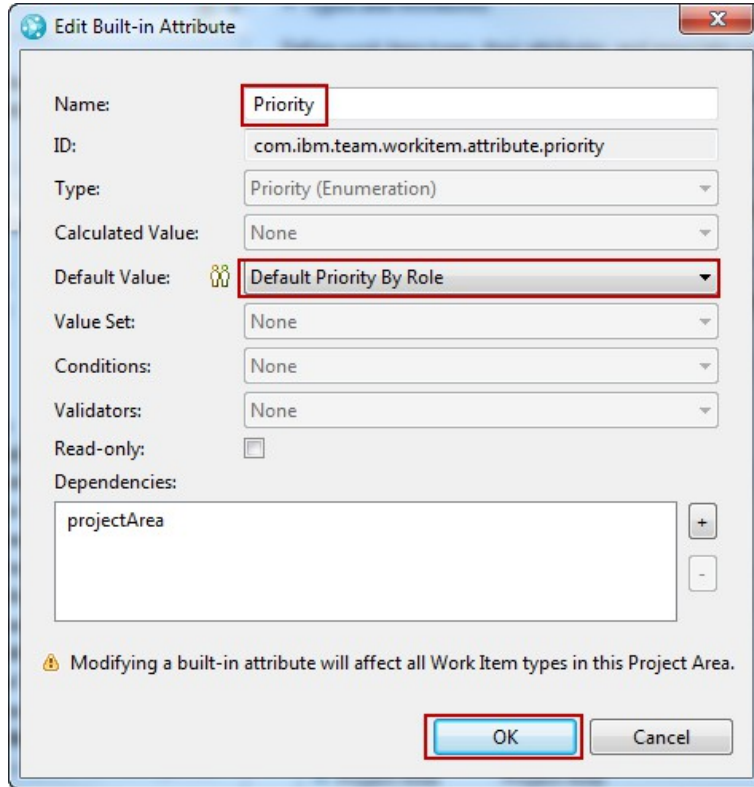
Configuration

Enumeration:

Role:	Literal:
<input type="text" value="Analyst"/>	<input type="text" value="Medium"/> ▼
<input type="text" value="Architect"/>	<input type="text" value="High"/> ▼
<input type="text" value="Developer"/>	<input type="text" value="Medium"/> ▼
<input type="text" value="Project Manager"/>	<input type="text" value="High"/> ▼
<input type="text" value="Stakeholder"/>	<input type="text" value="Low"/> ▼
<input type="text" value="Tester"/>	<input type="text" value="Low"/> ▼
<input type="text" value="default"/>	<input type="text" value="Unassigned"/> ▼

- __e. **Save** the process configuration with your changes.
- __3. Now the Attribute customization needs to be applied to the Priority attribute of the work items:
- __a. On the Process configuration editor navigate to the **Work Items > Types and Attributes** section.
- __b. Select a work item type that has the priority attribute. In our case for example the work item type *Defect*.
- __c. Scroll down to the Attributes section, browse for the Attribute named *Priority*. Right-click the row and select **Edit...**

- __d. On the Attribute Editor, the *Default Value* selection drop down button should be enabled. Click at the button and select the Attribute customization **Default Priority By Role**.

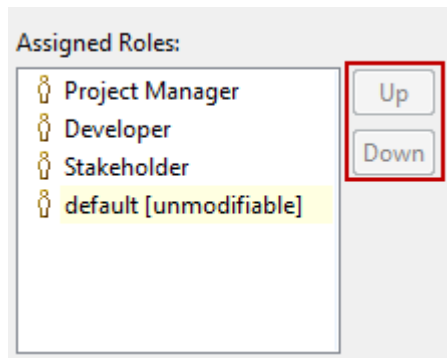


- __e. Select **OK**.
- __f. **Save** the changes to the process configuration.

__4. Test the default value provider

- __a. In the *Project Area* editor switch to the **Overview** tab.

In the *Members* section select the user **Jim** and click the **Process Roles...** button to see the process roles assigned to him. The user *Jim* has the role *Project Manager* as the primary assigned role:



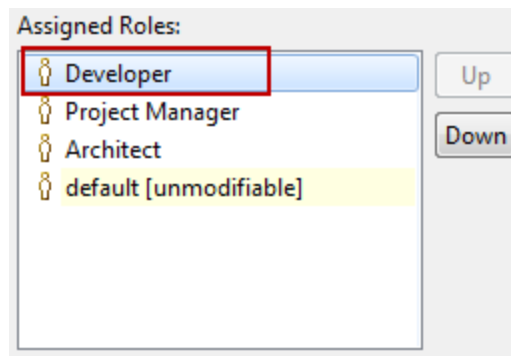
__b. Create a work item and check if the priority is set to the default value (High) expected for this role.

__5. Test how multiple roles work

__a. In the Project Area editor switch to the **Overview** tab.

__i. In the *Members* section select the user **Jim** and click the **Process Roles...** button to see the process roles.

__ii. Select the **Developer** role in the *Assigned Roles* column and use the **Up** button to move it to the top of the list.



__iii. **Save** the project area.

__b. Create a work item of type *Defect*.

__i. Check the priority.

The default priority is now set to the default for the Developer role (Medium) which is your primary role now.

__6. Change the roles setting for user Jim back to the original settings.

__a. In the *Members* section select the user **Jim** and click the **Process Roles...** button to see the process roles.

__b. Select the **Developer** role in the *Assigned Roles* column and use the **Down** button to move it to the second position of the list. See __4. __a. above for the desired setup.

__c. **Save** the changes to the Project Area Process configuration.

You have now successfully created and configured a role based default value provider. In addition you have observed how important the order of roles can be when configuring them in RTC.

Important RTC permissions and behavior lookup

For more information on how permissions lookup and the related operational behavior lookup works, see these articles which are still valid for RTC 4.x



- [Process permissions lookup in Rational Team Concert 2.0](#)
- [Process behavior lookup in Rational Team Concert 2.0](#)

Understanding these mechanisms is crucial for being able to manage process permissions and behavior in RTC.

4.1.3 Optional: Review the other available default value providers.



Section Scenario

Curious about which other default value providers are available, and how they work, you decide to have a closer look.

This section is optional and you can skip it.

- __1. Open the Eclipse Client if it is not already opened and connect to your repository. Log in with the user **jim** password **jim**. Open the *Project Area* editor for your Project.
- __2. Switch to the Project Area editor *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __3. Add a new Default Value in the *Attribute Customization* editor.
 - __a. Select **Default Value** in the editor. Use the **Add...** button at the bottom of the page to add a new Attribute Customization configuration.
 - __b. Select **Default Value** as *Type*.
 - __c. Verify you have the following *Providers* available:
 - __i. Boolean
 - __ii. Category
 - __iii. Duration
 - __iv. Enumeration List
 - __v. Iteration
 - __vi. Multi-Line HTML
 - __vii. Multi-Line Text
 - __viii. Number
 - __ix. Operating System
 - __x. Role Based Enumeration Default
 - __xi. Script Based Default
 - __xii. Single Line HTML


- __xiii. Single Line Text
- __xiv. User
- __xv. User List
- __xvi. Wiki

- __4. Create a Default Value provider for all other available types except Script Based Default. Note: We will look into script based value provider in a separate Lab.
 - __a. As name use the type name and append “default”. For example **Enumeration List Default**.
 - __b. For each of the default value providers check out the Configuration Editor section.

The screenshot shows a configuration window with two sections: 'Details' and 'Configuration'. The 'Details' section contains 'Name: Enumeration List Default' and 'Provider: Enumeration List'. The 'Configuration' section, which is highlighted with a red border, contains a dropdown menu labeled 'Enumeration:' and a label 'Enumeration List:' below it.

- For the typical default values it is possible to enter some data and in several cases also to pick valid values from the project area. List types allow to select multiple values.
- The role based enumeration default allows to configure default values based on the role of the user that creates the work item. See the next section.
- The type Operating System does not have an editor. To configure the values you need to edit the process XML source. See: https://jazz.net/wiki/bin/view/Main/AttributeCustomization#Operating_System_default_value. In addition it works only with the Eclipse client. You can however create your own enumeration for this purpose.

Limitations



There are some limitations for which default values are selectable for specific attribute types. For example multi Line HTML only works for Large HTML attributes and not for Medium HTML attributes.

4.1.4 Summary

The previous section showed how to create Default Value providers for RTC work item attributes. You have learned how to use the out of the box default value providers and successfully implemented some common examples.




There is a default value provider type called Script Based Default. This type will be handled with all other Script based value provider types in a separate Lab.

4.2 Value Sets

Sometimes it is necessary to be able to have more control about values that can be chosen for an attribute. RTC provides some options to do this by defining Value Sets.

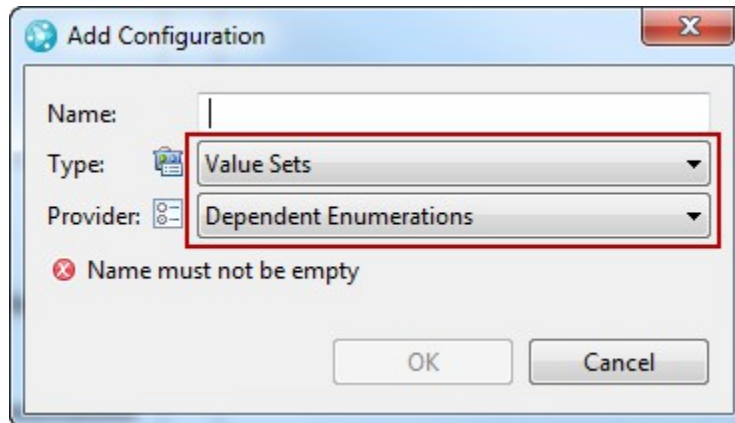
4.2.1 Dependent Enumerations

When working with several enumerations, sometimes certain combinations don't make sense. Often one attribute value limits which values should be selectable in another attribute. This can be done by creating Dependent Enumeration Value Sets. The setup of dependent enumerations is fairly straightforward.

	<p>Section Scenario</p>
	<p>Your project would like to provide the users with an easier way to choose from some of the attribute values available in the work items. They would like to provide the users with combinations of values that make sense in the context, to save the users having to think about it themselves.</p>
	<p>Because of your success so far, you have been elected to help the project with this task.</p>
	<p>The current process contains few enumerations to work with. Therefore the next steps are only meant to provide an overview.</p>
	<p>Suggested Reading If you need more examples please consider reading this article: https://jazz.net/library/article/537/</p>

- __1. Open the Eclipse Client if it is not already opened and connect to your repository. Log in with the user **jim** password **jim**. Open the *Project Area* editor for the *Nifty Application Project*.
- __2. Switch to the Project Area editor *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __3. Add a new Value Set in the Attribute Customization editor.
 - __a. Select *Value Sets* in the editor. Use the **Add...** button at the bottom of the page to add a new Attribute Customization configuration.
 - __i. For *Type* select *Value Sets* if it is not already selected.

- __ii. Select *Dependent Enumeration* for *Provider*.



- __iii. Name the configuration *Priority by Severity*. Select **OK**.

- __b. Configure the Value Set in the *Configuration* section of the editor to the right.

- __i. Select **Severity (Enumeration)** as *Source Attribute*.

- __ii. Select **Priority** as *Dependent Enumeration*

- __c. Now you need to map the Dependent Enumeration values to the Source Attribute Values.

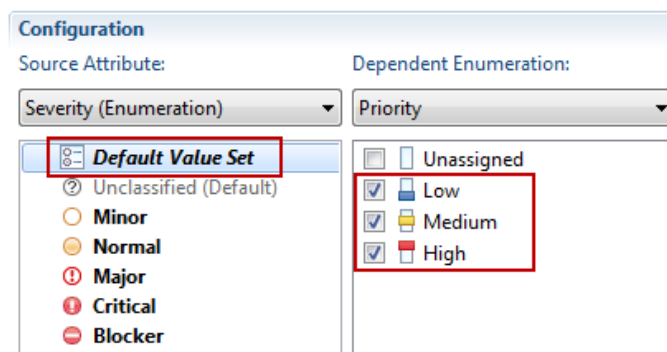
In this step you define which values on the right in the Dependent enumeration are applicable if the value on the left is selected.

This is just an example that shows the principle. We want to make sure that the priority for the work item is consistent with the severity of the work item.

- __i. Map the Severity value for the Default Value Set. The Default value set is the set of values that is allowed for the dependent enumeration for values of the source that are not specifically configured.

- __a. Select **Default Value Set** in the source attribute values.

- __b. Select the **Low, Medium, High** values as allowed in the Dependent Enumeration value list.



- __ii. Map the other Source Attribute Values
 - __a. Leave the source value *unclassified* **unspecified**
 - __b. Map the source value *Minor* to **Low** and **Medium**
 - __c. Leave the source value *Normal* **unspecified**
 - __d. Map the source value *Major* to **Medium** and **High**
 - __e. Map the source value *Critical* to **Medium** and **High**
 - __f. Map the source value *Blocker* to **High**
- __d. **Save** the changes to the process configuration.
- __4. Now configure the value set for the attribute Priority in the Types and Attributes Section.
 - __a. Open the **Work Items > Types and Attributes** section.
 - __b. Select the work item type *Defect* and scroll down to the *Attributes* section.
 - __c. Double-click the **Priority** attribute to open the *Attribute properties* editor.
 - __d. For *Value Set*, select **Priority by Severity**.
 - __e. In the *Dependencies* section, select “+” and add the *Severity* attribute. This is necessary to enable the Value Set provider and make it recalculate its value with the value of the severity attribute in case this changes.

Attribute Customization rely on dependent attributes



If a workitem customization such as a value provider depends on changes of other attributes, it is necessary to add the dependent attributes to the dependency list. If this is neglected, the value provider might not work as expected.

The attribute should look like below

The screenshot shows the 'Edit Built-in Attribute' dialog box with the following configuration:

- Name: Priority
- ID: com.ibm.team.workitem.attribute.priority
- Type: Priority (Enumeration)
- Calculated Value: None
- Default Value: Default Priority By Role
- Value Set: Priority By Severity (highlighted with a red box)
- Conditions: None
- Validators: None
- Read-only:
- Dependencies: projectArea, Severity (Severity is highlighted with a red box)

A warning message at the bottom reads: "Modifying a built-in attribute will affect all Work Item types in this Project Area." The 'OK' button is highlighted with a red box.

__f. Click **OK** and **Save** the changes you just did.

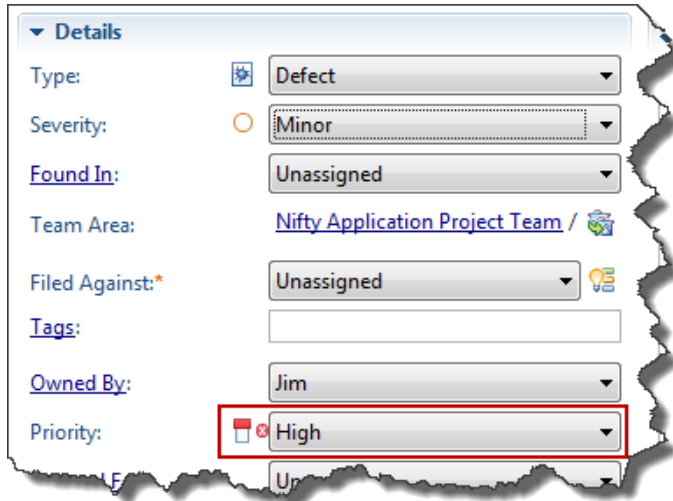
__5. Test your Attribute Customization

- __a. Open the overview tab of the project area editor and look at the user **jim** to check if the Project Manager role is the primary role. The first role in the process roles column is the primary role.
- __b. Create a *Defect* to test the change. As already mentioned, the customization affects the attribute across all work item types. So any work item type that has this attribute will be affected.

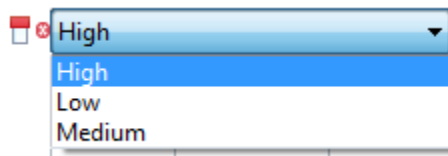
__c. Note that the default severity is *Normal*. The default priority is high due to your customization of the priority. The value is valid for our value set.

__i. Change the *Severity* to **Minor**

Note that an indicator in front of the priority now shows that the value does not conform your value set.



__ii. Change the *Priority* to *Low*.

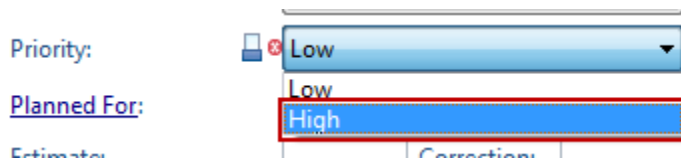


Click at the *Priority* attribute again. Now only *Medium* and *Low* are valid choices.

__iii. Change the *Severity* attribute to *Blocker*. The *Priority* attribute set to *Low* is flagged as invalid again.

__iv. Try changing the *Priority* attribute.

There is only one choice available due to your value set.: High. Low is only displayed because it is the current value. Choose *High* as the new *Priority*.



__d. Add the other required values and save the work item:

- __i. The save operation prevents you from saving the work item with a wrong enumeration value.

Save succeeds with RTC 4.0



Due to [defect 216723](#) RTC 4.0 allows saving values that are in violation with the Dependent Enumeration value set. This defect is fixed in 4.0.0.1.

You might also see unexpected validation error indicators upon saving the work item with RTC 4.0 as described in [defect 216173](#) which is also fixed in RTC 4.0.0.1.

- __e. Assign a valid value to save the work item in case the save fails or abandon saving the work item.

You have just successfully added a Dependent Enumeration value set.

4.2.2 HTTP Filtered Value Set

HTTP Filtered Value Sets allow you to provide values from external sources, typically from an XML Document that is published on a web server. The value set provider is very flexible and allows you to specify which data to extract from the XML Document using [XPath](#). A filter option allows you to handle longer value sets. Some post-processing allows you to compose data from several result columns.

Section Scenario



Your project now wants an attribute to support departments affected by a potential technology adoption. The attribute needs to be filled with some data from an external source.

Your project wants to use the data from another in-house application as the choice list in the work item attribute. The data is made available by a server in XML format.

You have been asked to help the project with this task.

Example Data

The HTTP Filtered Value Set requires a valid URL that provides XML information. It is not possible to use a local file and refer to it using the [file URI scheme](#) like *file://localhost/C:/temp/somefile.xml*.



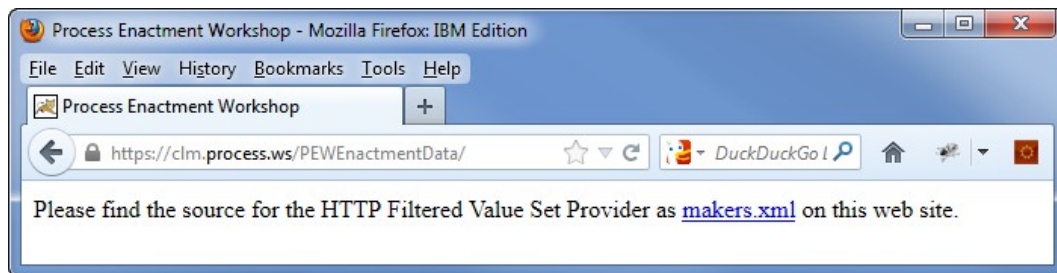
- For the Lab we use data that is provided using a custom WAR file that is deployed into the Jazz Team Server.
- You can also publish your own data easily using Apache. Install Apache as [described in this article](#) and place an XML document into the folder <Apache-Install_Dir>/htdocs. You can also provide a different folder as described in the article mentioned above.

- __1. Download the file *PEWEnactmentData.war* from the workshop on Jazz.net to *C:\JSWorkshops\Downloads* in case you have not already done so.
- __2. First deploy the XML data. To be able to deploy it on an application server such as Tomcat, the data is wrapped into a WAR file that can be deployed.
 - __a. Make sure the RTC Server is still running

- __b. Copy the file *PEWEnactmentData.war* from the download folder *C:\JSWorkshops\Downloads* into the folder *C:\JSWorkshops\IBM\JazzTeamServer\server\tomcat\webapps*.
- __c. Open the Tomcat console and make sure the file is deployed.

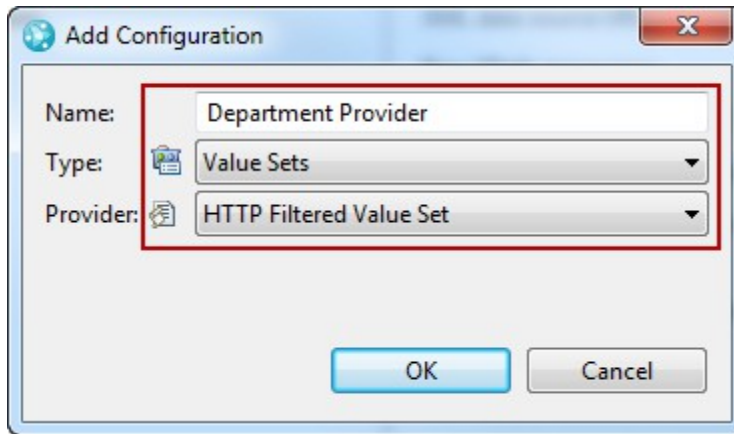


- __d. In your Browser type <https://clm.process.ws/PEWEnactmentData/> and make sure the browser shows the index page.



- __e. Follow the link to open the XML file.
- __3. Now open the Eclipse Client if it is not already opened and connect to your repository. Log in with the user **jim** password **jim**. Open the *Project Area* editor for your project.
 - __4. Switch to the Project Area editor Process Configuration tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __5. Add a new value provider in the *Attribute Customization* editor.
 - __a. Select *Value Sets* and use the **Add..** button to create a new configuration.
 - __i. For *Type* select *Value Sets* if it is not already selected.
 - __ii. Select *HTTP Filtered Value Set* as the *Provider*.

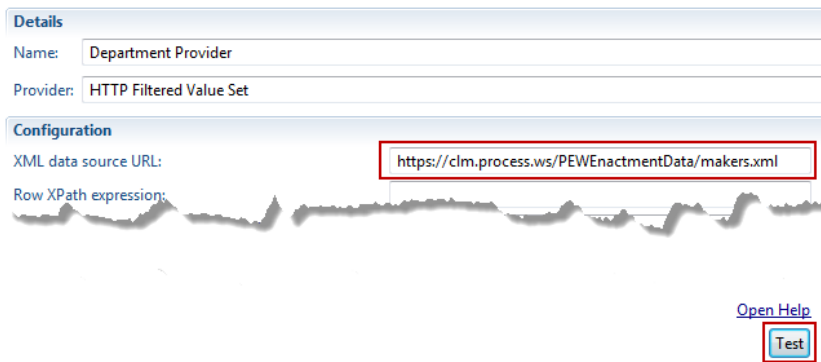
- __iii. As *Name* for the value set enter *Department Provider*. Select **OK**.



- __6. Now you need to configure the Value Set in the Configuration section of the editor.

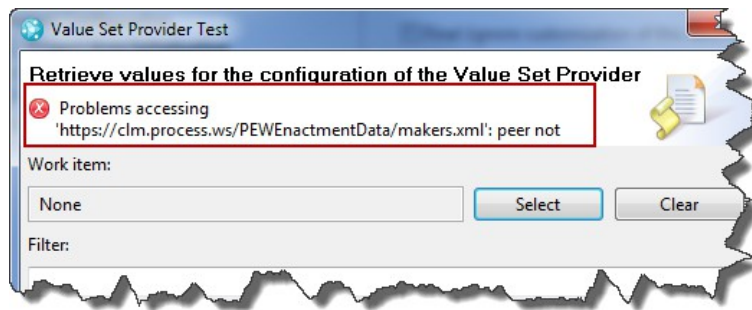
The basic steps are to provide a URL to get the data and XPath expressions to select the data from the XML source and define how the data is displayed and returned to the work item.

- __a. First you need to provide the URL to the data source. Enter "https://clm.process.ws/PEWEnactmentData/makers.xml" in the *XML data source URL* configuration property.
- __b. Maximize the editor window by double clicking on the tab, or scroll the configuration editor window to the lower right corner until you see the *Test* button.

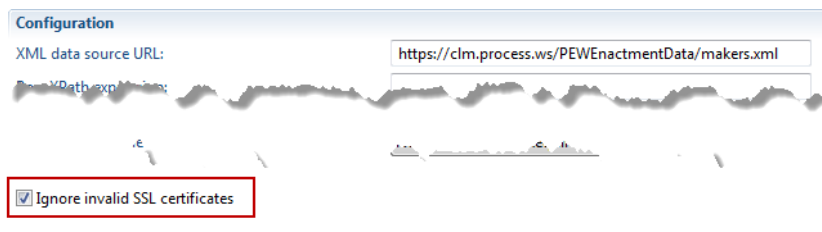


- __c. Press the **Test** button.

- __d. A window should come up. You will see an error. Hover over the error message to see the complete message displaying “*Problems accessing 'https://clm.process.ws/PEWEnactmentData/makers.xml': peer not authenticated*”

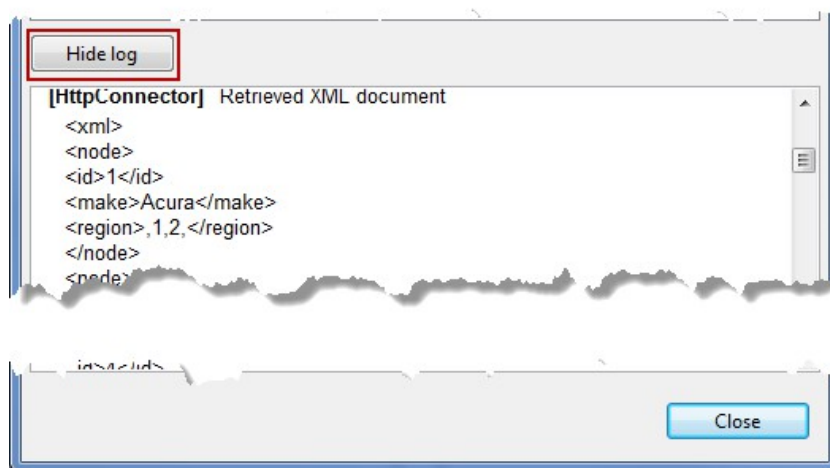
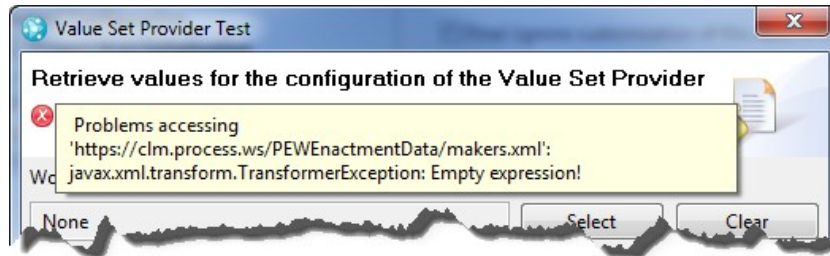


- __e. This error message is due to the default certificate being used with Tomcat.
- __f. Check the Ignore invalid SSL certificates at the lower left of the provider configuration and press the test button again.

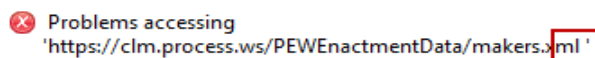


- __i. You should still see an error, because the configuration is not yet correct.

The output should look like below and using the *Show/Hide Log* button should provide you with the XML that was fetched if you scroll down a bit.




- __ii. If you see the error shown below and the log does not contain any XML, check the URL. You probably have a leading or trailing space in the URL. In production usage you might also want to check the Authentication settings and provide credentials, if required.



- __iii. If you can retrieve the XML, you can continue. Close the test window and save the process configuration.

- __7. In order to be able to use XPath to select the data from the XML file you need to understand the XML structure.

XPath tutorial and documentation



XPath is a language that allows you to navigate in XML. It allows to specify the path across the XML structure and to access elements and attributes. There are several tutorials available online. For example see <http://www.w3schools.com/xpath/> if you need to better understand how it works.

- __a. Open <https://clm.process.ws/PEWEnactmentData/makers.xml> in a browser. It should look like:

```

- <xml>
- <node>
  <id>1</id>
  <make>Acura</make>
  <region>,1,2,</region>
</node>
- <node>
  <id>2</id>
  <make>Aixam</make>
  <region>,2,</region>
</node>
<node>
<node>

```

- __8. [As described in Wikipedia, XPath](#) is a query language for selecting nodes from XML and to compute data from the XML data. From the above data we want to look at the data in the *node* entries.

The expression that selects the *node* data is `//xml/node`. It selects the nodes that matches the path selection `xml/node` in the document wherever they are. See http://www.w3schools.com/xpath/xpath_syntax.asp for more information.

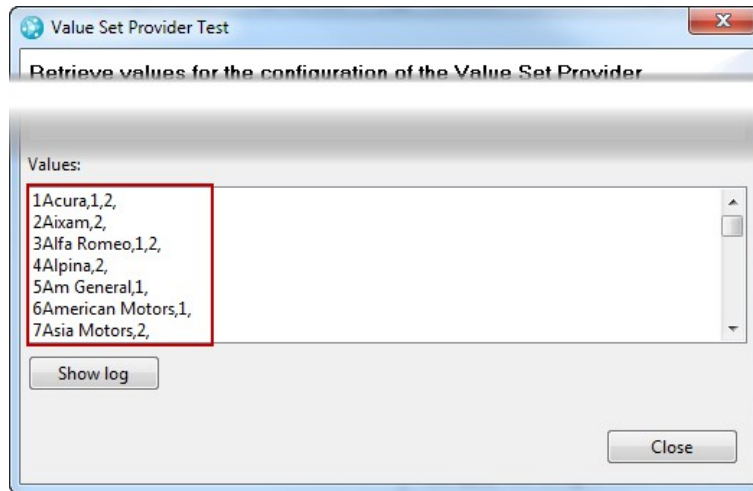
- __a. Return to the Process Configuration view in RTC. Enter `//xml/node` in the *Row XPath expression* configuration property.
- __b. Now it is necessary to select the data from the *node* elements. You want at least the data in the `<make>` element. Enter `./make` into the **Column XPath expressions** property.

Configuration

XML data source URL:	<code>https://clm.process.ws/PEWEnactmentData/makers.xml</code>
Row XPath expression:	<code>//xml/node</code>
Column XPath expressions (' ' delimited):	<code>./make</code>
Column identifiers (' ' delimited):	

- __c. Push the **Test** button in the lower right corner again.

You should now see some values



Troubleshooting




During creation of this workshop there were cases when the initial display did not work as expected. This seems to be related to the Column identifiers configuration property value. Please make sure to provide this value. Save the process configuration and try it again. The issue vanished after some tries.


- __d. Close the test window and **save** the process configuration.
- __9. The values displayed do not yet appear very user friendly. You want to work on the data to create a more usable value.
 - __a. As visible column identifier enter *Department* into the *Column identifiers* configuration property. This is used to determine how many columns are available.
 - __b. Currently we are only getting one column of data in a row. Enter *\${0} Department* into the *Entry label format* configuration property. This expression takes the column value, for example Acura, and adds the fixed string *Department* to it. This calculated string will be returned to the attribute making up our department names.
 - __c. **Check** the *Apply filter string to values received from the data source* option to allow users to reduce the choices.
 - __d. Leave *Sort values received from data source* **unchecked**.
 - __e. Leave *Ignore invalid SSL certificates* **checked**.

__f. You don't need to configure the *Authentication Method* for this example.


Ignore invalid SSL certificates

 In case the data source has a wrong SSL certificate this option allows you to ignore the certificate error and to retrieve the data despite the error.

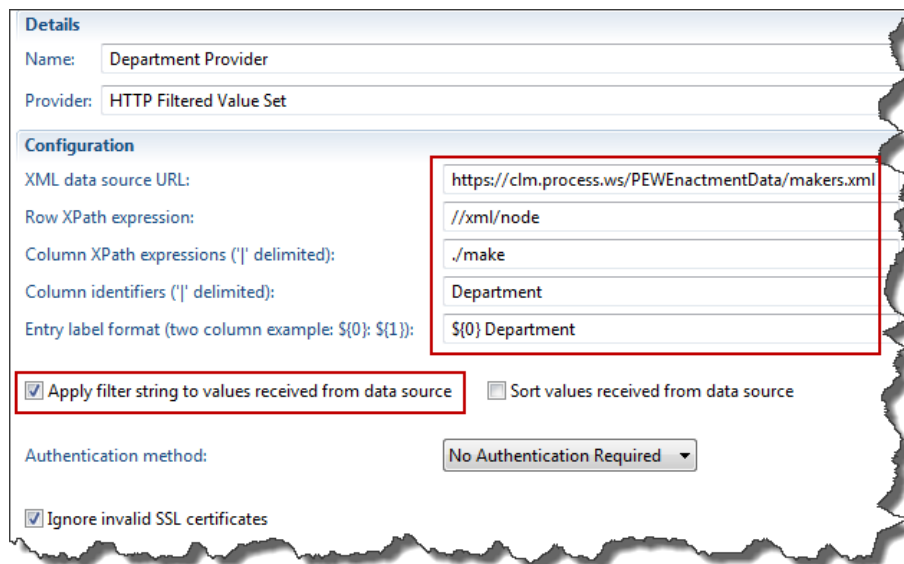
Sort values received from data source

 This option allows you to sort data sources that are not sorted before they are displayed.

Authentication Method

 The Authentication method can be used for data sources that require authentication. The value provider would access the site using the specified authentication method and the credentials.

__g. The configuration should now look like below.



Details

Name: Department Provider

Provider: HTTP Filtered Value Set

Configuration

XML data source URL:

Row XPath expression:

Column XPath expressions ('|' delimited):

Column identifiers ('|' delimited):

Entry label format (two column example: \${0}: \${1}):

Apply filter string to values received from data source Sort values received from data source

Authentication method:

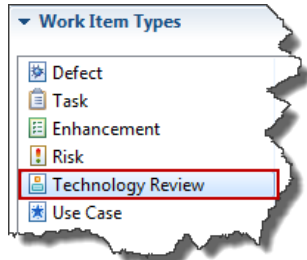
Ignore invalid SSL certificates

__h. Test the configuration using the **Test** button in the lower right corner of the Attribute customization editor again. You should now see a 'Department' postfix in the displayed values.

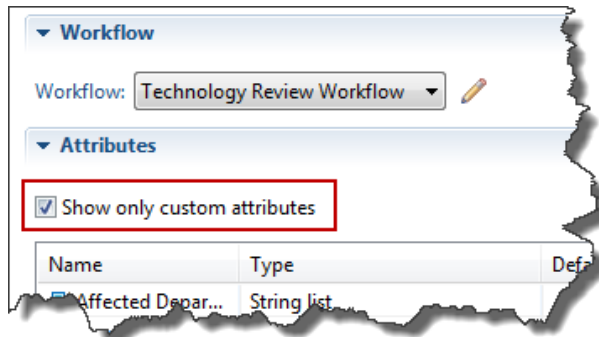
__i. **Save** the changes you just did to the process configuration.

__10. Now it is necessary to configure the attribute to use the HTTP filtered Value Set.

- ___a. Open the Project Area editor switch to the Process Configuration tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
- ___b. Select the *Technology Review* work item type. It is the only work item type with the Affected Department Attribute.



- ___c. Scroll down to the *Attributes* section. In the *Attributes* section **check** the “*Show only custom attributes*” checkbox to narrow down your search.



- ___d. Find the *Affected Departments* attribute and press the **Edit** button.

- __i. Use the drop down box button *Value Set* and select the *Department Provider* we just created. Select **OK**.

The screenshot shows a dialog box titled "Edit Custom Attribute". It contains several fields and dropdown menus:

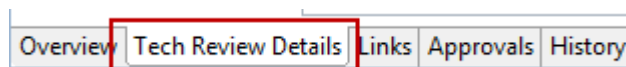
- Name: Affected Departments
- ID: com.acme.openup.workitem.attribute.affdepts
- Type: String list
- Calculated Value: None
- Default Value: None
- Value Set: Department Provider (highlighted with a red box)
- Conditions: None

At the bottom of the dialog are "OK" and "Cancel" buttons.

- __ii. **Save** the changes to the process configuration but leave the Project Editor open.

- __11. Test the HTTP Filtered Value Set you created.

- __a. Create a new work item of type *Technology Review*.
- __b. Switch to the *Tech Review Details* tab



- __c. In the *Adoption Cost Estimations* section look at the *Affected Departments* attribute.

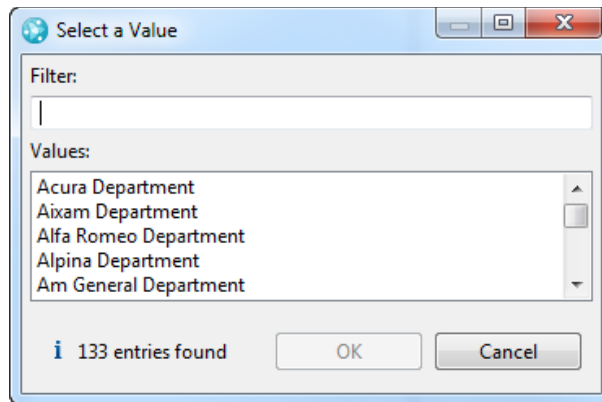
The screenshot shows a section titled "Affected Departments:" with a large empty text area below it. At the bottom of this section are three buttons: "New...", "Add..." (highlighted with a red box), and "Remove".



Troubleshooting

The attribute should show an Add... button. If the button does not show up, close the work item and repeat the steps from 10. Make sure to save the changes in the project area.

- __i. Press the **Add...** button and wait for the data to be retrieved. After a moment the *Select a Value* Window should show a window with a filter on top and some values below.



- __ii. Type **'I'** (capital 'i') into the filter. Your choices in the Values field should now be reduced to entries starting with an 'I'. Select the **International** Department.
- __d. Repeat the steps above and add the Mini Department.

__12. **Close** the work item, you don't have to save it.

You have just successfully configured your first HTTP Filtered Value Set.

4.2.3 Refine the HTTP Filtered Value Set



Section Scenario

You are not yet satisfied with the result. You want the department ID to be part of the value.

- __1. Open the *Project Area* editor for your Project.
- __2. Switch to the *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __a. Open the configuration editor for the Department Provider Value Set you just created. Expand the node *Value Set* and select the *Department Provider*.
 - __b. You want to modify the *Column XPath expression* to show an additional column. See the [XPath Syntax](#) for more information. You can add an additional column using the pipe symbol | and “.id” selects the ID in the XML. To make it more interesting we want to [use operators to calculate](#) a number using the ID.
 - __i. Enter the XPath expression *./make | (200-./id)* to the *Column XPath expression* property, to get two column values as result. The second column contains a value where the Id of the XML is subtracted from the constant value 200. The pipe symbol | separates the columns.
 - __ii. Add | *Identifier* into the *Column identifiers* property.
 - __iii. Add a *\${1}* statement to the *Entry label format property*, to insert the new column value in the front of the row.

__iv. Your configuration should look like below:

Configuration

XML data source URL:

Row XPath expression:

Column XPath expressions ('|' delimited):

Column identifiers ('|' delimited):

Entry label format (two column example: \${0}; \${1}):

Apply filter string to values received from data source Sort values received from data source

Authentication method:

Ignore invalid SSL certificates

__v. **Save** the process configuration changes.

__vi. Test the new configuration using the **Test** button at the bottom of the editor page.

__vii. The result now shows a number in front of the department name counting down from 200-1.

The list is no longer alphabetically sorted.

In addition using the filter no longer works as expected. Setting a filter 12 shows all entries starting with 12, but providing a character shows nothing.

Values:	
199	Acura Department
198	Aixam Department
197	Alfa Romeo Department
196	Alpina Department
195	Am General Department
194	American Motors Department
193	Asia Motors Department

__c. To get sorting back, close the test window and **check** the property '*Sort values received from data source*' to get it sorted again.

__i. Test the setting. Now it is sorted by the leading number.

__d. You want to get the filtering back, regardless how the output will finally look like and you would like to filter by department name. The filter value can be passed to the XML data source URL and the Row XPath expression as `${filter}` and used there.

- __i. In the *Row XPath expression* property enter `//xml/node[starts-with(/.make, '{filter}')]]`

This selects only the node elements that have a “make” element that starts with the string from the filter.

Clear the property 'Apply filter string to values received from data source'. Your data source is already filtered and leaving this checked would result in no output because the filter would filter the whole entry label.

Your Configuration should look like this now:

Configuration	
XML data source URL:	<input type="text" value="https://clm.process.ws/PEWEnactmentData/makers.xml"/>
Row XPath expression:	<input type="text" value="//xml/node[starts-with(/.make, '{filter}')]]"/>
Column XPath expressions (' ' delimited):	<input type="text" value="/.make (200-./id)"/>
Column identifiers (' ' delimited):	<input type="text" value="Department Identifier"/>
Entry label format (two column example: \${0}: \${1}):	<input type="text" value="\${1} \${0} Department"/>
<input type="checkbox"/> Apply filter string to values received from data source	<input checked="" type="checkbox"/> Sort values received from data source
Authentication method:	<input type="text" value="No Authentication Required"/>
<input checked="" type="checkbox"/> Ignore invalid SSL certificates	

- __ii. Test the value set provider and enter 'I' (capital 'i') as filter. The values displayed should only show departments starting with 'I'. If you type a small 'i' character, the result will be empty. Upper or lower case makes a difference.

Ignore Case Filter



With XPath 1.0 there seems to be no way to use the filter with ignore case. It also does not support wildcard characters. It would however be possible to use the XPath expression `contains()` or other options as [described here](#) instead of `starts-with()`.

- __iii. **Close** the test window.
- __e. You finally decide you like the department number but would rather have it at the end of the string and not modified.
- __i. Change the *Entry Label format* property to `${0} Department (${1})`
- __ii. Remove the calculation and pass the original ID by changing the *Column XPath expressions* property to `./make | ./id`

__iii. Leave all other settings. The configuration should look like

Configuration

XML data source URL:

Row XPath expression:

Column XPath expressions ('|' delimited):

Column identifiers ('|' delimited):

Entry label format (two column example: \${0}: \${1}):

Apply filter string to values received from data source Sort values received from data source

Authentication method:

Ignore invalid SSL certificates

__f. Use the **Test** button to test if the configuration works. You should get something like the following

Value Set Provider Test

Retrieve values for the configuration of the Value Set Provider

Select a work item, optionally enter a filter string and validate the retrieved items

Work item:

Filter:

Values:

- AM General Department (5)
- Acura Department (1)
- Aixam Department (2)
- Alfa Romeo Department (3)
- Alpina Department (4)
- Americal Motors Department (6)
- Asia Motors Department (7)

__3. **Close** the test window and **save** the process changes to the project area to keep your settings.

You have successfully created a HTTP based filtered value set provider.

4.2.4 Additional Information for HTTP based filtered value providers

There is more you can do with the filtered value provider. You can pass the value and the label of any attribute of the current work item in the *XML data source URL* and the *Row XPath expression*. See the [help topic](#) for more details.

If the XML data source accepts more parameters, it would be possible to pass an attribute value, for example a technology name, using another custom attribute of the Technology Review work item type. For example this string could be passed:

```
http://cars.flashmx.us/makes?${com.acme.openup.workitem.attribute.techname}
```

The Authentication for the HTTP based filtered value provider can be configured and supports OAuth. This allows you to use any OSLC source that can be contacted and is correctly set up. For example, you can use a friend relationship as XML data source. See more details here:

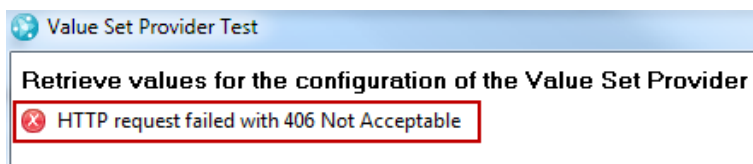
<http://jorgediazblog.wordpress.com/2012/06/27/work-item-customization-httpconnector-and-oauth-in-rtc-4-0-for-oslc/>

Accept Statement is required





The HTTP based filtered value posts a specific *Accept: application/xml* statement in the request header. If the web server exposes an XML page but does not accept the request header you will see an error. See below.

For example http://www.w3schools.com/xml/cd_catalog.xml returns



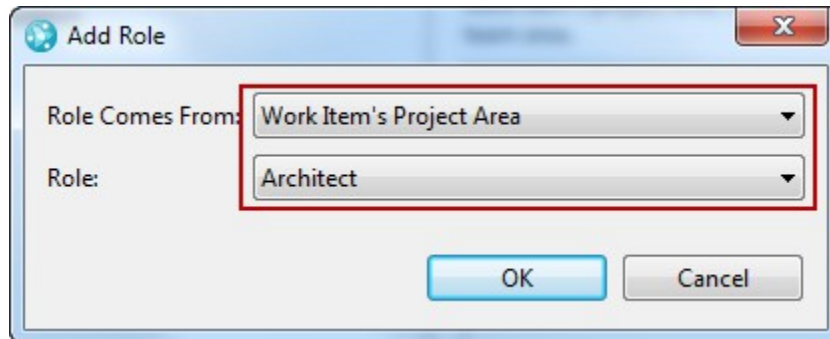
4.2.5 Role Based User List

It is sometimes interesting to be able to have additional attributes that store one or more users that have a special relationship to a work item. This can be done by creating attributes of the type “contributor” or “contributor list”. Sometimes it is convenient or required to have a limited list of users to choose from. This is what the role based user list helps with.

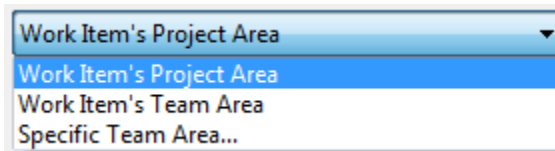
	<p>Section Scenario</p>
	<p>Your team has decided that it is desirable to assign an architect to be responsible for each Technology Review work item.</p> <p>You have been asked to help the project with this task.</p>
	<p>Please note that the Value Set Provider created in this lab only works for attributes of type Contributor. It does not work for attributes with the type Contributor List. Enhancement 222235 requests this capability.</p>

- __1. Open the Eclipse Client, if it is not already opened, and connect to your repository. Log in with the user **jim** password **jim**. Open the *Project Area* editor for your Project.
- __2. Switch to the Project Area editor *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __3. Select *Value Sets* in the editor. Use the **Add...** button at the bottom of the page to add a new Attribute Customization configuration.
 - __a. For *Name* enter *Responsible Architect*.
 - __b. For *Type* make sure to select *Value Sets*.
 - __c. As *Provider* select *Role Based User List*.
 - __d. Press the **OK** button.
 - __e. Now configure the new Value Set.

- __i. In the configuration editor press the **Add Role...** button to add a role configuration. In the configuration dialog you can select a process area where the role comes from and the role to select the user.



- __ii. For the configuration *Role Comes From* choose *Work Item's Project Area*.



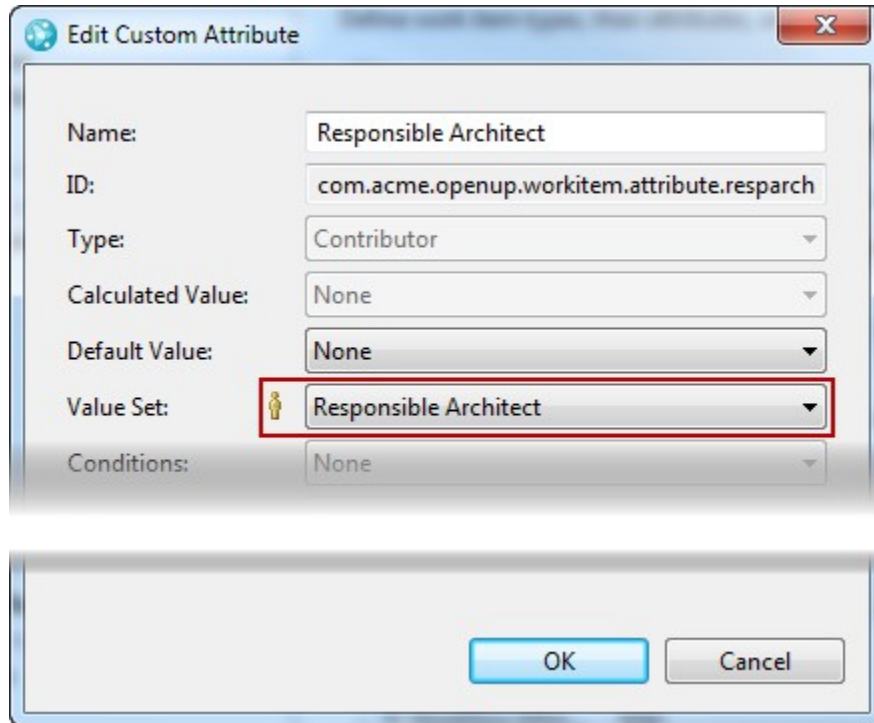
Roles Available In The Selection



Please note: If you choose the “Work Item's Project Area” or “Specific Team Area...” only the roles defined in the chosen process area are selectable. If you select “Work Item's Team Area” all roles, including roles added in different Team Areas, are selectable. Note also that selectable roles for the “Specific Team Area...” option includes the roles inherited from parent process area as well.

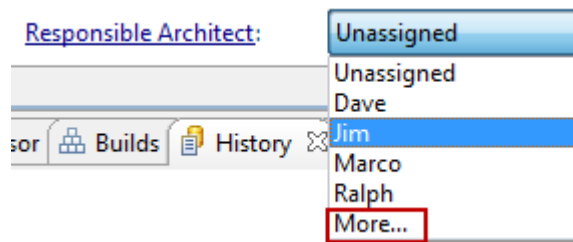
- __iii. From the available roles select *Architect*.
 - __iv. Click **OK**.
 - __v. It is possible to add additional configurations for where the role comes from and which role to choose by repeating the previous steps.
 - __f. **Save** the changes to the process configuration.
- __4. Now it is necessary to configure the attribute to use the Role Based User List. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
- __a. Select the *Technology Review* work item type. Scroll down to the *Attributes* section.

- __b. In the *Attributes* section check the **Show only custom attributes** checkbox.
 - __i. Select the custom attribute *Responsible Architect* and click **Edit...**
 - __ii. In the *Custom Attribute* editor for the Responsible Architect attribute select the *Responsible Architect Role Based User List* using the *Value Set* drop down button.



- __iii. Click **OK** and **save** the process changes to the process configuration.
- __5. Check the project area Members and make sure that one or more members have the Architect role. If you add some Architect Roles to users, make sure to save the changes before you continue.
- __6. Open a work item of type *Technology Review*. If you have such a work item already open close it first and re-open it to make sure the value set is available to the editor.
 - __a. Open the *Tech Review Details* tab of the work item.

- __b. Click on the *Responsible Architect* attribute. The drop down list should show the users available in the project area with the role Architect.



- __c. You can still assign other users using the **More...** button at the end of the drop down.

- __7. You don't have to save the work item at this time.

You have just successfully implemented a Role Based User List!

4.2.6 Summary

In the previous section you learned how to create Value Sets for RTC work item attributes. This allows to reduce the number of selectable values and select from external data and makes your process more streamlined for your requirements.

There is a value set provider type called Script Based Value Set. This type will be handled with all other Script based value provider types in a separate Lab.

4.3 Validators

Section Scenario

Your team maintains a block of effort and cost estimation values in the Technology Review work item type. Your team wants to track and calculate some values to estimate total cost for certain technologies that are reviewed.



Currently a user can type anything into these values and there is no validation of the format available. The attributes are realized as String to give more flexibility, but it is probably tough to get a reasonable number value out of any possible string, such as “Foo * Bar + Server”. The team is nervous about that and would like to make sure the data is correct and consistent.

Since you have been so successful in implementing all the other requests, your project has learned to rely on you. They ask you to solve the issue and implement the solution. So you head to your cubicle to find out what options you have.

It is a typical requirement that work item attributes can only have certain values. Specially for attributes of string types it is often necessary to validate that the value follows some pattern.

Currently the following attributes are maintained in the Technology Review work Items. These attributes are supposed to be used with an automated calculation.

Estimated Effort:	<input type="text"/>
Average Cost:	<input type="text"/>
Estimated Investment:	<input type="text"/>
Total Cost:	<input type="text"/>

The way this is supposed to work is someone enters an effort estimation and an average cost for the Technology adoption effort. The Estimated Investment is used to track infrastructure cost the company has to put into adopting the new technology. So, ultimately you want to be able to show:

$$\text{Total Cost} = \text{Estimated Effort} * \text{Average Cost} + \text{Estimated Investment}$$

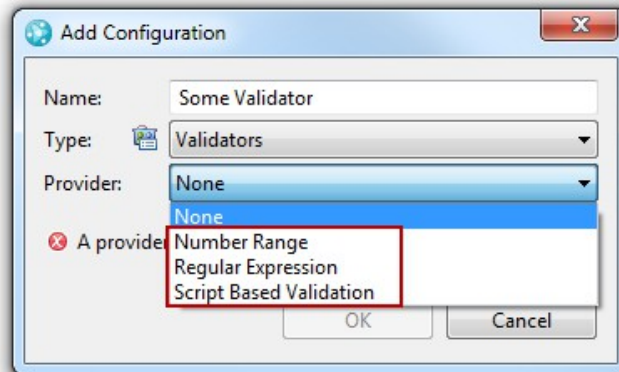
Rational Team Concert provides validators to validate attribute values and make sure they conform to a special format.

- ___3. Open the Eclipse Client if it is not already opened and connect to your repository. Log in with the user **jim** password **jim**. Open the *Project Area* editor for the *Nifty Application Project*.
- ___4. Switch to the *Project Area* editor *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.

__5. Use the **Add...** button to add a new Attribute Customization.

__a. Select *Validators* as the *Type*.

__b. Inspect the configuration property *Provider*. You can choose from Number Range, Regular Expression and Script based Validation.



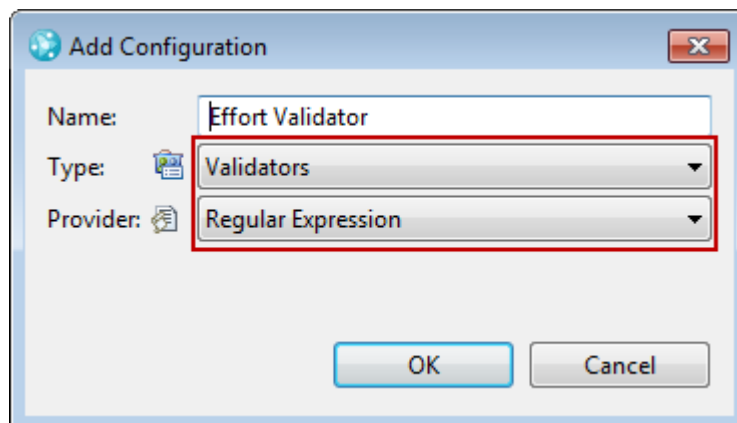
__i. Basically you could try using a number range as validator.

The number range checks if a number is in a certain range.

Unfortunately this validator works only for number types and is therefore not usable in the current example where all attributes are of a string type.

So you need to select *Regular Expression* as the *Provider*. The regular expression works with string types.

__c. Change the *Name* to *Effort Validator* and click **OK**.



__d. On the configuration editor you can now configure your regular expression. For some information on regular expressions have a look at this [Wikipedia link](#). It refers to additional tutorials.

__i. Choose an *Error* in the *Decoration Icon* property.

- __ii. You want to accept input like **1d** for one day and we want to support weeks, months etc. In the *Message* configuration property enter

Format: Any number optionally followed by m, w, d, h for month, weeks, days...

- __iii. In the *Regular Expression* configuration property enter

`(\d*m)|(\d*w)|(\d*d)|(\d*h)|(\d*)`

It validates to true for inputs with a number prefix and allow to input m, w, d, h or leave the suffix out.

- __iv. Check the **Case sensitive** checkbox. We only want to accept small characters.

Details

Name: Effort Validator

Provider: Regular Expression

Configuration

Decoration Icon: Error

Message: Any number optionally followed by m, w, d, h for month, weeks, days...

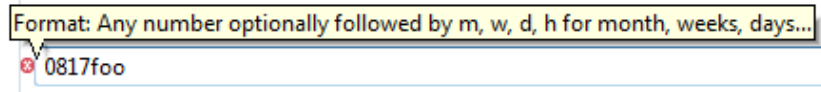
Regular Expression: `(\d*m)|(\d*w)|(\d*d)|(\d*h)|(\d*)`

Case sensitive

Match line breaks (for ^ and \$)

Try the Regular Expression on sample text:

- __a. Enter some samples in the sample text input to test your expression and validate that you get an error for values that are wrong.



- __b. If you hover over the error marker you get to see the description.

- __v. **Save** the changes to the process configuration.

- __e. Create another Validator for the cost attributes.

- __i. Enter *Cost Validator* as *Name*.
- __ii. Choose *Validator* as *Type*.
- __iii. Use the *Regular Expression* as *Provider*.

- __f. Configure the validator.
 - __i. Choose **Error** in the *Decoration Icon* property.
 - __ii. You want to accept input like 1M for one million, 100T for 100.000 and any number. In the *Message* configuration property enter

Format: 1M, 4T, any number...
 - __iii. In the *Regular Expression* configuration property enter a pattern similar to the one you already used

`(\d*M)|(\d*T)|(\d*)`
 - __iv. Make sure the **Case sensitive** checkbox is selected.
 - __v. Test the validator by typing in some valid and invalid values.

Details

Name:

Provider:

Configuration

Decoration Icon:

Message:

Regular Expression:

Case sensitive

Match line breaks (for ^ and \$)

Try the Regular Expression on sample text:

- __g. **Save** the process change to the Project Area process configuration.
- __6. To configure the attributes, open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section in the *Process Configuration* tab of the Project Area editor.
 - __a. Select the *Technology Review* work item type, scroll down to the *Attributes* section, select the **Show only custom attributes** check box.

- __i. Double click at the *Estimated Effort* attribute to open the Custom Attribute Editor. Use the *Validators* selection drop down button, select the *Effort Validator*. Then select **OK**.

The screenshot shows the 'Edit Custom Attribute' dialog box. The 'Name' field contains 'Estimated Effort'. The 'Value Set' dropdown is set to 'None'. The 'Conditions' dropdown is set to 'None'. The 'Validators' dropdown is set to 'Effort Validator', which is highlighted with a red box. The 'Read-only' checkbox is unchecked. At the bottom, there are 'OK' and 'Cancel' buttons.

- __b. Double click at the *Cost* attribute to open the Custom Attribute Editor. Use the *Validators* selection drop down button, select the *Cost Validator*. Select **OK**.
- __c. Double click at the *Estimated Investment* attribute to open the Custom Attribute Editor. Use the *Validators* selection drop down button, select the *Cost Validator*. Select **OK**.

Name	Validators
Affected Departments	
Complexity	
Cost	Cost Validator
Estimated Effort	Effort Validator
Estimated Investment	Cost Validator
Impact	

- __d. **Save** the process change.
- __7. Test your Attribute Customization
- __a. Create a work item of type Technology Review. You have to close and re-open the work item, if you want to use an existing one.
- __b. Enter the summary.
- __c. Go to the Tech Review Details tab.
- __i. Enter some text that does not match the valid formats

__ii. An error shows up in front of the attributes.

Estimated Effort: foo

Average Cost: bar

Estimated Investment: foo bar


Total Cost:

Overview Tech Review Details Links Approvals History

__iii. Press the **Save** button.

__d. The save happens despite of the errors.

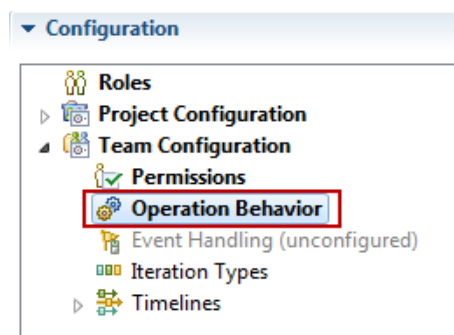
The Validator does not prevent the work item from being saved



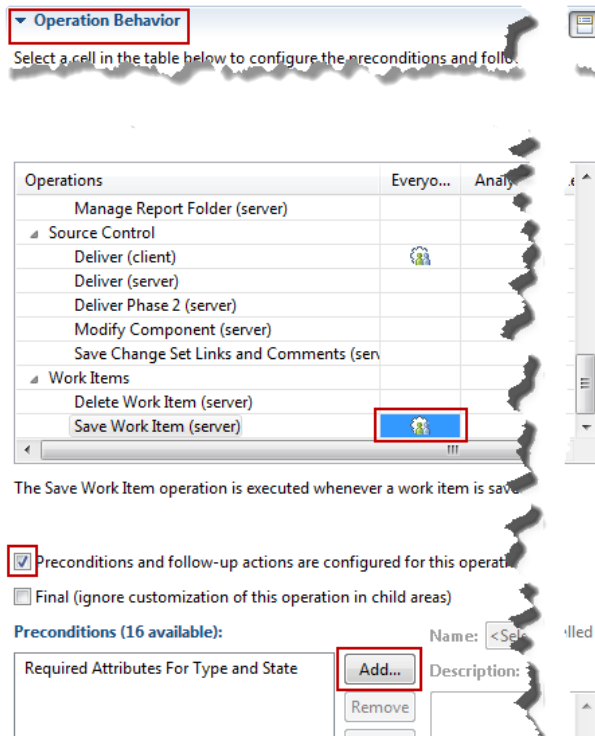
The validation itself does not prevent the work item from being saved. If you want a validation error to prevent the work item from being saved, you have to configure a precondition for the validation and the validations have to return an error. Enablement of this precondition is showed in following steps.

__8. Configure the precondition to prevent the work item from saving in case the validation of an attribute fails.

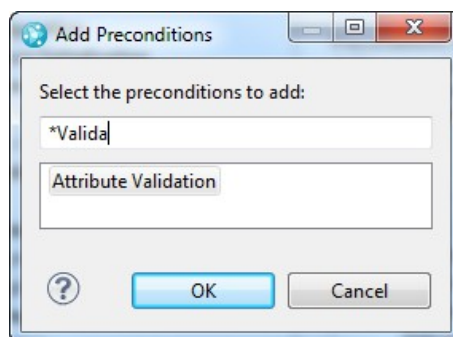
__a. On the *Process Configuration* tab of the *Project Area* editor open the **Configuration > Team Configuration > Operation Behavior** section.



- __b. In the *Operation Behavior* editor section, scroll down and find the *Work Items > Save Work Item (server)* operations for the role *Everyone* in the related table column.



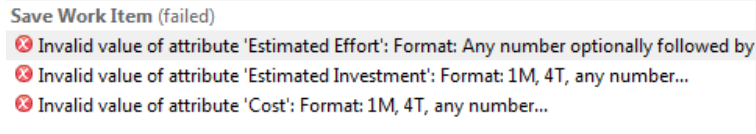
- __i. In the column for the role *Everyone* click the element in the **Save Work Item (server)** row.
- __ii. Make sure the check box in front of “*Preconditions and follow-up actions are configured for this operation*” is **checked** and press the **Add...** button.
- __iii. Enter **Valida* in the search field to find the *Attribute Validation* precondition.



- __iv. Select the *Attribute Validation* precondition and press **OK**.
- __v. You can specify a different name and a more descriptive error message text if you like. Other than that there are no configuration options.
- __vi. **Save** the process configuration change to the project area.

- __c. Go back to the work item that has the validation errors.
 - __i. Do a small change for example in the *Summary* attribute. Press the **Save** button again.

Now the save operation fails with an error. This happens in the clients as well as in the Web UI.



- __ii. Modify the values for the attributes *Estimated Effort*, *Estimated Investment* and *Cost* to values that match the desired formats. For example use 20d, 10T and 1M as input. Press the Save button.
 - __iii. Now the work item saves without errors
- __9. Try other decoration icons such as Warning in your validator.

- __a. Open the Project Area editor switch to the Process Configuration tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.

- __i. Expand the *Validators* node in the editor.
- __ii. Open the *Effort Validator*. Switch the Decoration Icon setting from *Error* to *Warning*.
- __iii. Save the changed process configuration.


- __b. Switch to your work item.

If using the Web UI close the work item editor and open it again, so that the UI is able to recognize the change.


- __i. Type a wrong value into the Estimated Effort attribute. Notice the warning symbol showing up.
- __ii. Press the **Save** button.
- __iii. The save proceeds without an error.

This is due to the Warning level.


Only validations of level error prevent from saving!

 The Attribute Validation precondition only prevents saving for validations that are set to level error.


The Attribute Validation precondition is global!

 The Attribute Validation precondition, if configured, will work for all work item types that have the attribute. Use other levels such as warning or information if the precondition is configured.

Operation Behavior can be complex!

 Operation behavior is configured by role and what is configured at which role will define which rules apply. For example operational behavior does not accumulate across user roles. E.g. the operational behavior configured for the highest role that is found will be taken.

Important information!

 Please carefully read the articles about

- [Process behavior lookup](#) and
- [Process permissions lookup](#)

in RTC to understand which rules apply. The articles are for RTC 2.x but the content is still valid.

- __c. Open the *Effort Validator* again. Switch the Decoration Icon setting back from Warning to Error.
- __d. Switch the *Effort Validator* back to *Error* level.
- __e. **Save** the process configuration.

4.3.1 Summary

In the previous section you learned how to create attribute validators for RTC work item attributes and how to prevent a work item from saving in case a validation fails.

There is a Validation provider type called Script Based Validation. This type will be handled with all other Script based value provider types in a separate Lab.

4.4 Summary

In this lab you learned how to use Attribute Customization to make using work items easier and enforce consistence of values or formats. To do so you learned how to:

- Create Default Value Providers
- Create Calculated Value Providers
- Create Value Sets
- Create Validators and how to prevent saving invalid values

Lab 5 Work Item Customization With JavaScript

This lab will step you through some advanced work item attribute customization provided by Rational Team Concert.

Rational Team Concert work items provide a rich set of attribute types and values for different needs. These needs may change as your process matures. The last lab showed how you can easily customize work items and values using declarative customization available in RTC. Sometimes the built in customization options still don't provide the desired capabilities. This lab will show how JavaScript Based Attribute Customization can add even more flexibility by:

- Providing default values based on the context of values of other attributes
- Calculating attribute values based on other attributes values
- Validating attribute values based on other attribute values
- Providing attribute values available for selection based on context
- Making attribute values required or optional based on context
- Making attribute values read only based on context

Lab Scenario



You are helping your team to customize a Rational Team Concert process. Your team requires more complex customization. Since you are tasked to do all customization for the project team, you will learn about JavaScript based customization such as:

- Default Values for attributes
- Calculated Values and value sets for attributes
- Conditions for attributes
- Validators for attributes that make sure the format of the data is correct when saving a work item.

Suggested Reading



For more information please read

- [Getting Started with Work Items in Rational Team Concert](#)
- [Customizing attributes in Rational Team Concert 4.0](#)
- [Customizing attributes in Rational Team Concert 3.0](#)
- [Wiki entry: Work Items attribute customization](#)

5.1 Introduction to JavaScript Based Attribute Customization

Section Scenario



You realize that some of your project's requirements can not be implemented with the declarative work item customization you used in the previous lab. There is a feature called JavaScript based Customization and you are wondering if that could be a solution for the project's more complex requirements. Since it looks like your early successes has put you in the role as the one and only process customization expert, you decide to learn what this script based customization is all about.

In the first section you will:

- Learn about some capabilities and limitations of JavaScript based Customization.
- Enable your server to use JavaScript based Customization
- Optionally install a JavaScript development environment on your RTC client.

The easier customization options you have seen only go so far. IT is sometimes desirable to perform more complex tasks such as calculating across complex dependencies and values. Rational Team Concert supports capabilities for implementing more complex behavior in several ways. It is possible to extend the Rational Team Concert Server and the Web UI using Java and JavaScript (taking advantage of the Dojo Toolkit). These kinds of extensions can get quite complex and require a lot of knowledge about the API.

How can RTC be extended?



Please read the article [Extending Rational Team Concert 3.x](#) as a good entry point about options to extend RTC. [The Rational Team Concert 4.0 Extensions Workshop](#) is a newer resource than the one referenced in the article above.

RTC has the capability to use a simpler approach based on JavaScript to create Attribute Customization. This capability has been in the product since version 3.x

If using RTC 4.0.3 or greater



Rational Team Concert versions 4.0.3 and greater have differences in the Script Based Configuration Editor for Attribute Customization from the documented ones in this lab. Check out the information in [Appendix D – Changes in 4.0.3 attribute customization script based editor](#) for guidelines in how to follow the lab using the new editor features.

5.1.1 JavaScript Based Attribute Customization Capabilities and Limitations



Theory section

This section contains some background information about Attribute Customization Capabilities and Limitations. It is content for reading and can be skipped if necessary.



Suggested Reading for JavaScript Based Attribute Customization

Please read the article [Work Items attribute customization - Using scripts for attribute customization](#) in addition to following this workshop.

The JavaScript based Attribute Customization allows more flexibility than the attribute customization you have seen so far.

The JavaScript based Attribute Customization currently supports:

- Reading work item attribute values
- Writing work item attribute values
- Computing with work item attribute values and other data

JavaScript based Attribute Customization allows you to work with the [following Supported Attribute Types](#):

- Short String
- Medium String
- Large String
- Integer
- Long
- Big Decimal
- Boolean
- Timestamp [as an ISO-8601 standard string](#).

The SDK Package **com.ibm.team.workitem.api.common** shows the portion of the API that is exposed to JavaScript. See the [RTC Extensions Workshop on how to install the SDK](#) if you want to have a deeper look. The following operations on work item attributes are currently supported:

- `workItem.getValue(attributeID)` to get the value of the attribute
- `workItem.getLabel(attributeID)` to get labels and names for attribute values
- `workItem.isAttributeSet(attributeID)` to determine if the attribute is available

The capability to access work item attribute labels is the method `getLabel()`, which represents the display value of the work item attribute value for most of the types is new in RTC 4.0.

Enumeration Types: It is possible to access enumeration types. The operation `getValue()` returns the literal ID of the Enumeration Literal. The operation `getLabel()` allows to access the Enumeration Literal display value. To set an enumeration value use the Literal ID. See [Working with Enumerations](#) for more information about using Enumerations in scripts.

There is only **limited support for Items**.

- It is possible to read an item type attribute, but this only returns the item ID.
- A new capability in RTC 4.0 is the ability to access work item Item attribute value labels using `getLabel()`. In RTC 4.0 labels for attribute types state, category, project/team area, contributor attributes are supported.
- It is possible to set Item type attributes, such as owners, using the ID values that are returned reading these types of attributes in a calculated or default value.
- There is currently no way to access additional information for the item from the ID.

While it is possible to read the work item state, **it is not possible to detect if the state changed**. Only the current (or new) state of the work item can be accessed.

It is not possible to access the relationships of a work item, especially to other work items from the links tab. It is possible to access work item attributes of type work item. This will return the work item handle but will currently not allow to resolve the referenced work item.

The current limitations constrain what can be done using JavaScript. If you run into them you most likely have to create a real extension. However, the feature is still useful in a lot of situations.

5.1.2 Challenges Developing JavaScript Based Attribute Customization



Theory section


This section contains some background information about Challenges Developing JavaScript Based Attribute Customization. It is content for reading and can be skipped if necessary.

Developing JavaScript based Attribute Customization is not a trivial task. There are various challenges:

- The JavaScript editors are typically not able to provide as much automation and support as other editors. The editors can detect less syntax errors and do not provide code completion to the same extent provided by other editors. This often results in failing scripts with no obvious reasons. Since JavaScript is not as strict as Java, script failures don't necessarily provide clear indications about what caused the error.
- JavaScript does not provide the same capabilities Java provides.
- Searching the internet seems to provide fewer examples and good documentation compared to other languages such as Java.
- Debugging the scripts: only scripts running on the client (attribute customization for validators or calculated values), can be debugged with tools like [Firebug](#). See [this information on debugging scripts](#) for complete information. [Appendix B – Script debugging](#) shows an example of how to use that debugger with one of the sample scripts used in this lab.

Since debugging is only possible for certain customization, and the error messages thrown by JavaScript code errors are sometimes not meaningful, we will discuss script code logging as a development approach for script development and code debugging. See the Trouble Shooting Appendix on page 192 for some hints and the referred [Appendix B](#) for example of alternate debugging client code.

5.1.3 Enable JavaScript



Enable JavaScript

The JavaScript based attribute customization needs to be enabled on the CCM server before it can be used.

- __1. Open a browser and navigate to <https://clm.process.ws/ccm/admin> you need to log in with a user that has JazzAdmin repository permission. You can use user ID *pewadmin* password *pewadmin*.
 - __a. Navigate to the Advanced Properties <https://clm.process.ws/ccm/admin?action=com.ibm.team.repository.admin.configureAdvanced> .
 - __b. Search for the property “**Enable Process Attachment Scripts**”. Set the value to **true**.

com.ibm.team.workitem.service.internal.WorkItemRepositoryService	
Property	Current Value
Enable Process Attachment Scripts	true
Maximum Attachment Size (MB)	50

- __c. **Save** the change that you just did
- __2. You can close the browser.

You have now successfully enabled using JavaScript based work item customization.

5.1.4 Optional: Install the Web, XML, and Java EE Development Tools



Add JavaScript Editor to Eclipse

You can use any preferred editor to develop JavaScript. If you want to use the RTC Eclipse client and have installed the vanilla RTC Eclipse client, it is possible to install the **Web, XML and Java EE Development** extension from the update site for your Eclipse version.

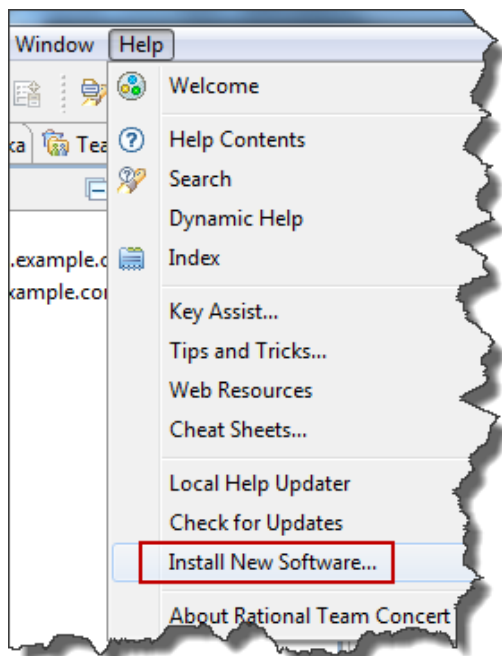
Make sure to install the right version!



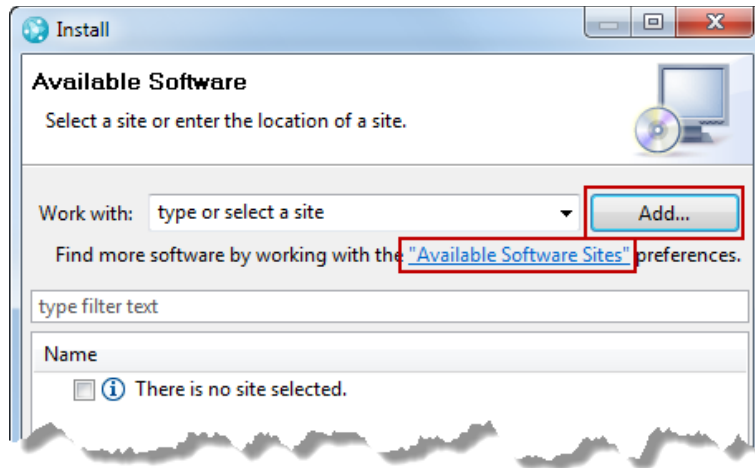
Make sure you understand which version of Eclipse you have installed and that you install a matching version of the extension. Information about which Eclipse version you have installed can be found in Help>About Rational Team Concert one of the Eclipse icons



- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user **jim** password **jim**.
- __2. Open the Eclipse **Help** menu and select **Install New software...**



- __a. If you have not configured any update sites press the **Add...** button, otherwise use the Available Software Sites for your Eclipse client and continue in step

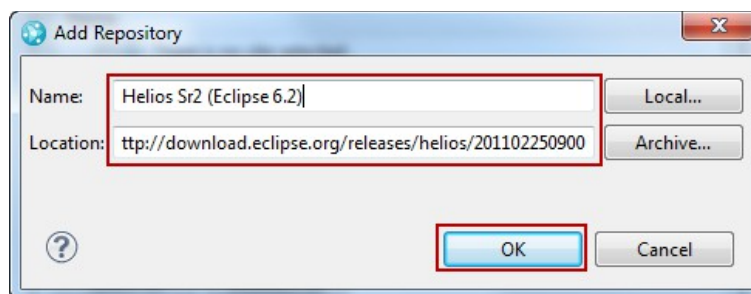


- __i. In the dialog name the Software Site for example Helios Sr2 (Eclipse 6.2)

Update Site depends on RTC Client Eclipse Version

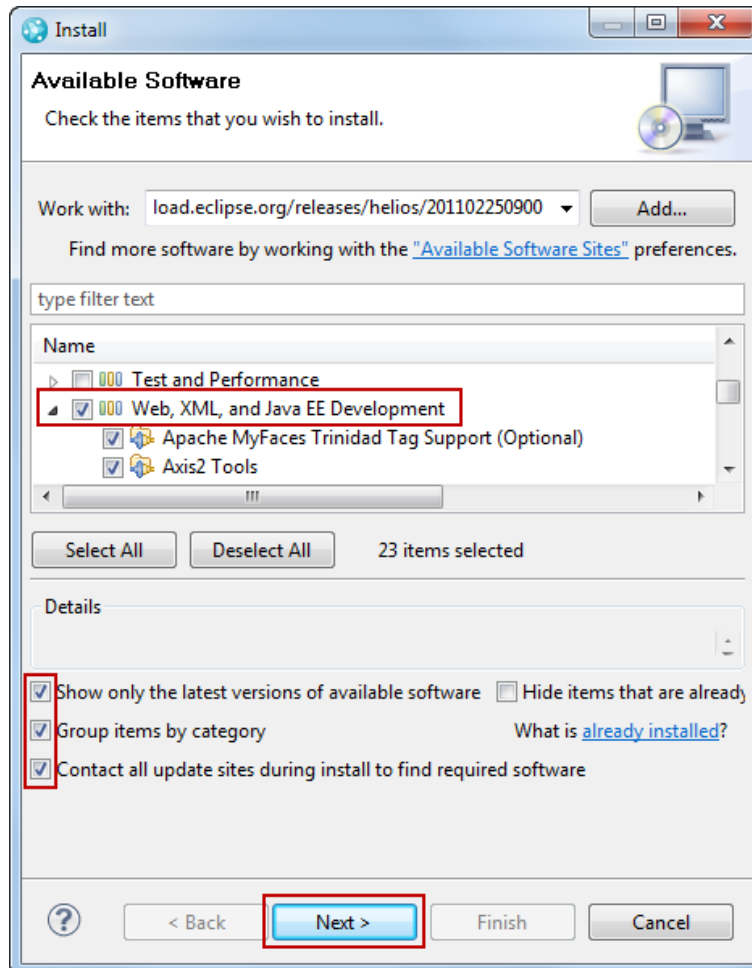
Make sure to verify if the update site you are using is compatible to your RTC Eclipse client.

- __i. Enter a valid Software Site URL for the version you are using.
<http://download.eclipse.org/releases/helios/201102250900> works for a vanilla RTC 4.0 zip install based on Eclipse 3.6.



- __ii. Press **OK** to continue.
- __b. Eclipse will read the content on the selected update site
- __c. Group the items by category and select the items to install the Web, XML, and Java EE Development category, which includes the JavaScript editor.

__d. Press Next.



__e. Eclipse will calculate the required dependencies and display what it is going to install. Press **Next**.

__f. Accept the license and press **Finish**.

__g. Restart your RTC Eclipse client when prompted.

You are now able to create projects for JavaScript development and to create and edit JavaScript source files in your RTC Eclipse client.

5.2 Script Based Calculated Value

In some cases the value of an attribute is really derived from a calculation with the value of one or many other attributes. Rational Team Concert uses some built in Calculated Values available for the Formal Project Management Template. It also allows to create Calculated Values using JavaScript. In the next section you will learn how to create JavaScript based Calculated Values,

5.2.1 Total Cost Calculated Value

Section Scenario



Your project wants to calculate the total cost for adopting a technology from the estimated effort, the average cost per hour, and the estimated investment. The way this is supposed to work is someone enters an effort estimation and an average cost for the Technology adoption effort. The Estimated Investment is used to track infrastructure cost the company must put into adopting the new technology.

Ultimately you want to show *Total Cost = Estimated Effort * Average Cost + Estimated Investment*.

You will use a Script Based Calculated Value for this.

Example for Script Debugging

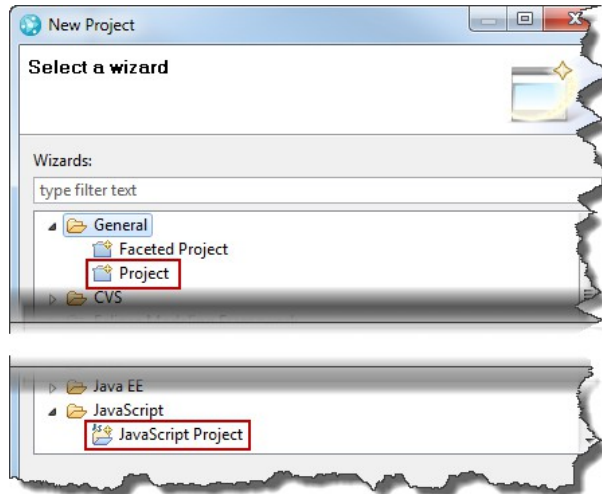


The script in this section is the one used for the debugging using Firebug example in [Appendix B](#) section.

- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user **jim** password **jim**.
- __2. Create a new Eclipse project.

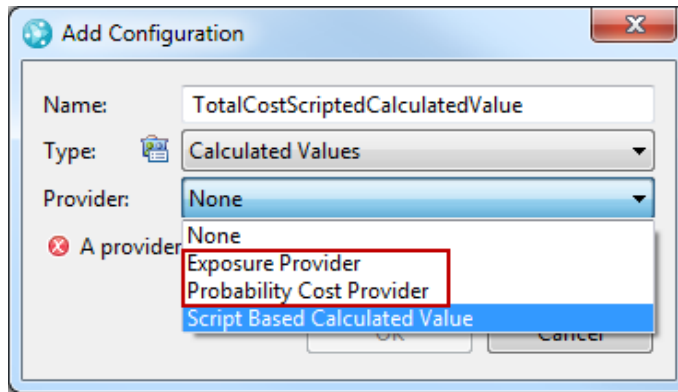
- __a. Use the Eclipse menu **File > New > Project** to create a project.

If you installed the JavaScript support in 5.1.4 create a JavaScript project, otherwise create a general Project.



- __b. Name the Project *ScriptBasedCustomization*
- __c. Leave the defaults and press **Finish** to create the project.
- __i. If you create a JavaScript Project you can dismiss to open the JavaScript perspective.
- __d. Use the menu **Window > Show View > Other** and open the *Project Explorer* view to see the new project.
- __3. Go to the *Team Artifacts* view.
- __4. Right click at the *Nifty Application Project* project area in the *Team Artifacts* view and select **Open** to open the *Project Area Editor* for your project.
- __5. Switch to the Process Configuration tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __a. Use the **Add...** button to add a Calculated Value
- __b. As Name enter *TotalCostScriptedCalculatedValue*
- __c. Select the *Type Calculated Values*

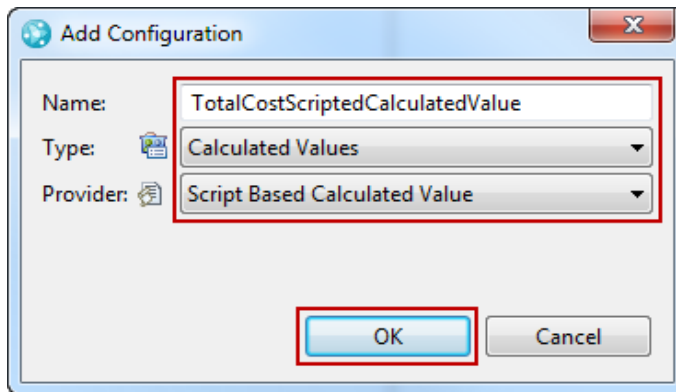
__d. Review the available **Providers** by selecting the drop down box..



The Exposure Provider and the Probability Cost Provider are built in providers for the Formal Project Management Template. In your context they are not applicable.

__i. Select **Script Based Calculated Value** as *Provider*.

The configuration dialog should look like below:



__ii. Press **OK** to set the configuration.

__e. **Save** the change to the process configuration.

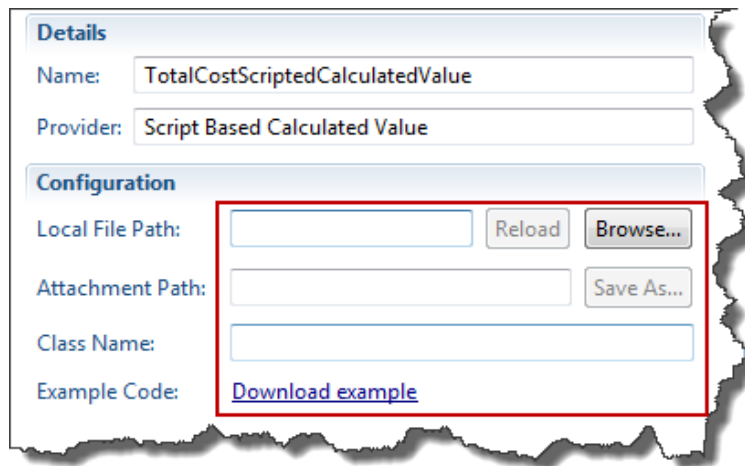
__6. Inspect the *Configuration* editor section to the right.

If using RTC 4.0.3 or greater



Rational Team Concert versions 4.0.3 and greater have differences in the Configuration editor from the documented ones in this lab. Check out the information in [Appendix D – Changes in 4.0.3 attribute customization script based editor](#) for guidelines in how to follow the lab using the new editor features.

- __a. There is a *Local File Path* property to browse for a local script file and to reload the script file after changes.
- __b. There is an *Attachment Path* property describing where the file will be stored in the process configuration.
- __c. There is a *Class Name* property to specify the name of the class that is called in the script.



The screenshot shows a configuration editor with two main sections: **Details** and **Configuration**. The **Details** section contains a **Name** field with the value "TotalCostScriptedCalculatedValue" and a **Provider** field with the value "Script Based Calculated Value". The **Configuration** section contains a **Local File Path** field with a **Reload** button and a **Browse...** button, an **Attachment Path** field with a **Save As...** button, a **Class Name** field, and an **Example Code** field with a [Download example](#) link. A red box highlights the **Configuration** section.

- __d. Finally there is a link to download example code.
- __7. Download the example code into your project.
 - __a. Click at the **Download example** link in the Configuration editor.

If using RTC 4.0.3 or greater

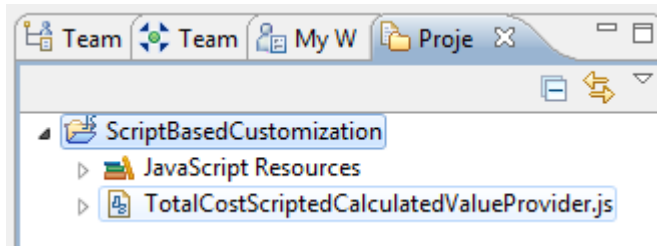


You will find instead a "Fill in example" link. Once clicked you can use "Save As..." to download the script sample as instructed in this lab, or perform the script changes using the new in place script editor.

See [Appendix D – Changes in 4.0.3 attribute customization script based editor](#) for more information.

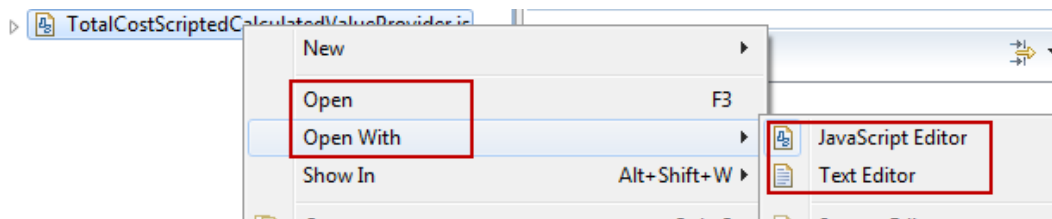
- __b. Browse to the folder `C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\`
- __c. Change the file name to `TotalCostScriptedCalculatedValueProvider.js`
- __d. Press the **Save** button.
- __e. **Save** the process configuration.
- __8. To see the new script in Eclipse open the *Project Explorer* view.

- __a. Select the *ScriptBasedCustomization* project
- __b. Use the *Refresh* context menu entry or press *F5*.
- __c. The externally saved script file now shows up in Eclipse



__9. Inspect the script example

- __a. Open the file in Eclipse by double clicking or using open or open with.



Using JavaScript Editors



You can use a JavaScript editor or the Text editor. The JavaScript editor is syntax aware and makes working with JavaScripts easier.

You can also open the scripts using an external editor. In the Eclipse Preferences you can configure Eclipse to use your preferred editor with the **.js** file type either using an internal or external editor.

__b. Either way, the example should look similar to the code below

```

⊕ * Licensed Materials - Property of IBM
dojo.provide("com.example.ValueProvider");

(function() {
    dojo.declare("com.example.ValueProvider", null, {
⊖
        getValue: function(attribute, workItem, configuration) {

            return workItem.getValue(attribute);

        }
    });
})();

```

__c. To rename the value provider class name replace the string "com.example.ValueProvider" in both the *dojo.provide* and *dojo.declare* statements with the string "**com.acme.providers.script.TotalCostScriptedCalculatedValue**"

__d. **Save** the change.

__10. The example currently does nothing except passing the current value of the attribute through. To make sure you can see it working, you want to provide some debug output in the logs.

__i. In the line before the return statement add the line

```
console.log("- Start");
```

__ii. Your script should now look as below.

```

dojo.provide("com.acme.providers.script.TotalCostScriptedCalculatedValue");

(function() {
    dojo.declare("com.acme.providers.script.TotalCostScriptedCalculatedValue", null, {
        getValue: function(attribute, workItem, configuration) {
            console.log("- Start");
            return workItem.getValue(attribute);
        }
    });
})();

```

__iii. **Save** your change.

__11. Switch over to the Project Area editor. The configuration should look like below:

Details

Name:

Provider:

Configuration

Local File Path:

Attachment Path:

Class Name:

Example Code: [Download example](#)

Using RTC 4.0.3 or greater screenshot doesn't match



Remember the information in [Appendix D – Changes in 4.0.3 attribute customization script based editor](#). If using RTC 4.0.3 or greater the screenshot from above will not match your environment.

- __a. Check the file name in the *Local File Path* is correct set to *C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\TotalCostScriptedCalculatedValueProvider.js*
- __b. Check the *Class Name*. It shows the old name *com.example.ValueProvider*. This is wrong since our change.
- __c. Press the **Reload** button. You might have to maximize the editor to see all buttons. Double click at the editors tab to maximize. Double click the maximized tab to minimize.

- __d. Confirm the *Class Name* now shows *com.acme.providers.script.TotalCostScriptedCalculatedValue*

Details	
Name:	TotalCostScriptedCalculatedValue
Provider:	Script Based Calculated Value
Configuration	
Local File Path:	C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\TotalCostScrip <input type="button" value="Reload"/> <input type="button" value="Browse..."/>
Attachment Path:	/workitem/scripts/common/TotalCostScriptedCalculatedValueProvider.js <input type="button" value="Save As..."/>
Class Name:	com.acme.providers.script.TotalCostScriptedCalculatedValue
Example Code:	Download example

- __e. **Save** your changes to the process configuration.

__12. Configure the attribute to use the JavaScript Based Calculated Value.

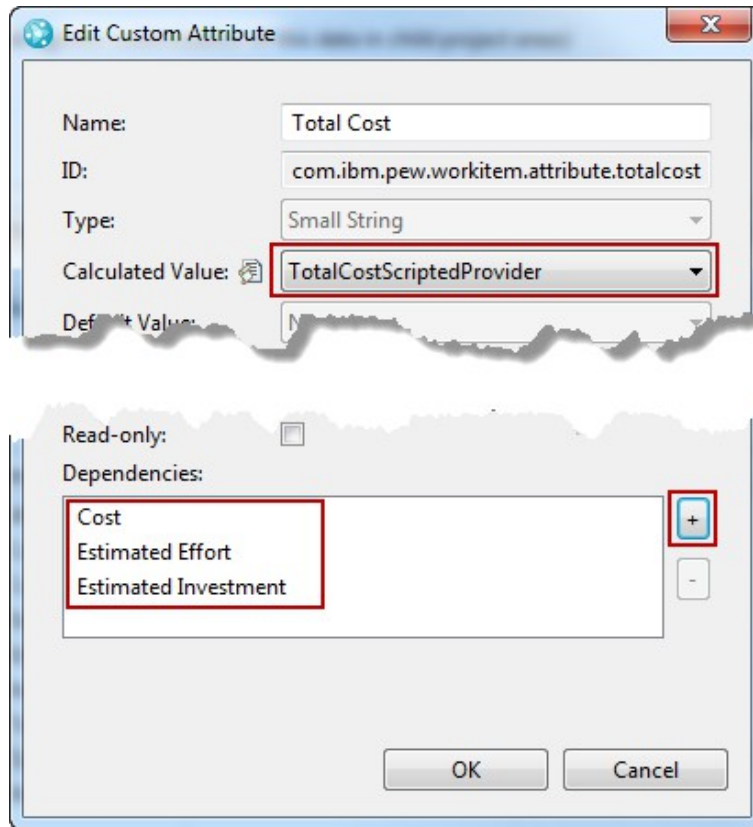
- __a. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
- __b. Select the *Technology Review* work item type. Scroll down to the *Attributes* section.
- __c. In the Attributes section check the **Show only custom attributes** check-box to narrow down your search.
- __i. Locate and select the custom attribute *Total Cost*. Press the **Edit** button.
- __ii. Use the drop down button *Calculated Value* to select the **TotalCostScriptedProvider**.
- __iii. Add dependencies to the attributes **Cost**, **Estimated Effort** and **Estimated Investment**. Every time one of these attributes change, the new calculated value provider needs to get called to do the calculation based on the new data.

Attribute Customization rely on dependent attributes



If a workitem customization such as a value provider depends on changes of other attributes, it is necessary to add the dependent attributes to the dependency list. If this is neglected, the value provider will not be triggered by changes in the dependent attributes and might not work as expected.

Your Custom Attribute Configuration should now look like below.




__d. Click **OK** and **save** the changes to the process configuration.

- __13. Now that the prototype Calculated Value provider is configured, you can test if it runs. Log files provided by the products will help with testing.

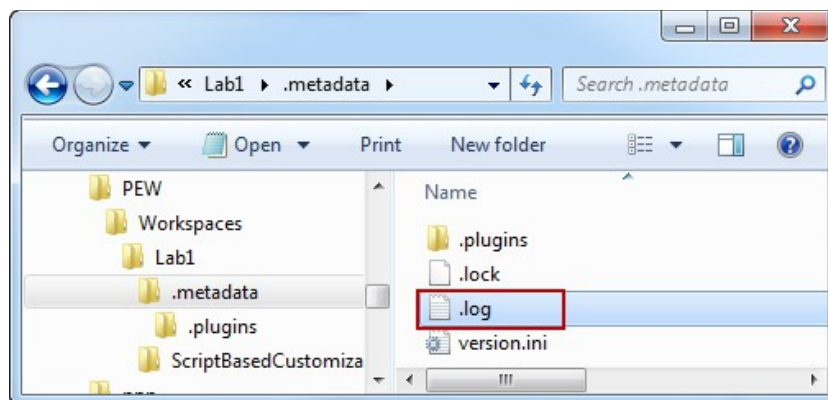
Log Files for JavaScript Based Customization

See this [information about logging and debugging](#).

For the Eclipse client the log file is located in the .metadata folder in the workspace folder and called .log.
For example
C:\JSWorkshops\Workspaces\Lab1\.metadata\.log

 For the Web UI the log data is located in the server log file for example
<JazzServerInstallDir>\server\tomcat\work\Catalina\ocalhost\ccm\eclipse\workspace\.metadata\.log
or
<JazzServerInstallDir>\server\tomcat\work\Catalina\ocalhost\jazz\eclipse\workspace\.metadata\.log
if you have a 2.x context root.
In this workshop <JazzServerInstallDir> is represented by C:\JSWorkshops\IBM\JazzTeamServer

- __a. Test if the script works in Eclipse using the log file:
- __i. Use the Eclipse client to create a new work item of type *Technology Review*.
 - __ii. Open an explorer window, locate the Eclipse log file C:\JSWorkshops\Workspaces\Lab1\.metadata\.log and open it with a text editor



Use more advanced text editors



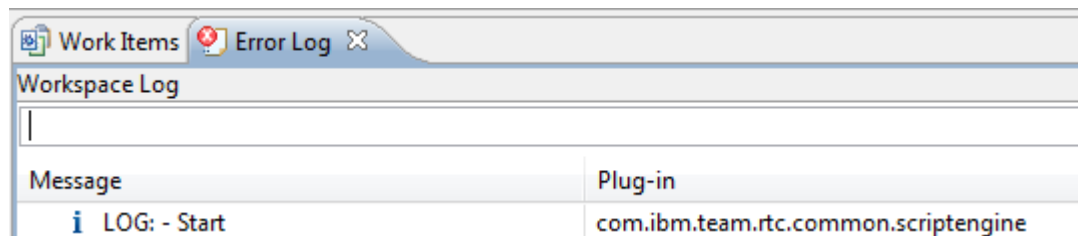
When using more advanced text editors such as [Notepad++](#) you can keep the log file loaded and just let it reload if a change is detected. It also highlights several occurrences of the same text in several lines which makes analyzing log files easier.

Advanced editors also often support JavaScript syntax highlighting.

- __iii. Scroll to the end of the log file. It should show a log entry of the script engine and the log message like below.

```
!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:24:36.455
!MESSAGE LOG: - Start
```

- __b. (Optional) Inspect the *Error Log* view. You can also integrate the log view in your eclipse client for this debugging exercise by opening the Error Log view: **Window > Show View > Other > Error Log**. You should be able to see something like this:



Troubleshooting



If there is no such entry, check if scripting is enabled, check the configuration of the attribute was done correctly and try to save the scrip, reload it and save the process configuration again. Close and reopen the client and create a new work item of type Technology Review

- __c. Test the script works in the Web UI.
- __i. Use the Web UI <https://clm.process.ws/ccm/web> to create a new work item of type *Technology Review*.
- __ii. Open the log file
 C:\JSWorkshops\IBM\JazzTeamServer\server\tomcat\work\Catalina\localhost\ccm\eclipse\workspace\metadata\log

- __iii. Scroll to the end of the log file. It should show a log entry of the script engine and the log message like below.

```
!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:26:43.365
!MESSAGE LOG: - Start
```

- __d. If the log does not show this information, the script did not run and something is wrong. See the Troubleshooting Box above.
- __i. Check if scripting is enabled.
- __ii. Close the work items and create a new work item of type Technology Review.
- __iii. Reload the script after fixing and save the process configuration.

__14. Currently the script does not do anything, you want to add some useful code.

- __a. First it is necessary to read the attributes. Then the calculation needs to be performed.
- __b. To be able to read attribute values the script needs access to some API.
- __i. Add the dojo.require statement below

```
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");
```

```
⊕ * Licensed Materials - Property of IBM
dojo.provide("com.acme.providers.script.TotalCostScriptedCalculatedValue");
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");
```

- __c. Now it is possible to access the workitems attributes through the referenced JavaScript API.

Access to attributes is provided by *workitem.getValue(AttributeIDString)*. It is necessary to pass the Attribute ID for the work item. The API provides access to the built in attributes providing access to the [ID's from WorkItemAttributes](#) class.

Accessing built-in attributes of the work item

The examples above use the `com.ibm.team.workitem.api.common.WorkItemAttributes` class to access built-in attributes of work items. Here are all possible values:

- `WorkItemAttributes.SEVERITY`
- `WorkItemAttributes.PRIORITY`
- `WorkItemAttributes.FOUND_IN`
- `WorkItemAttributes.ID`

Read Custom Attributes

WorkItemAttributes only defines the built in attributes.



For Custom Attributes, you have to look up the Attribute ID in the process configuration, for example by clicking at the attribute in the Types and Attributes section.

If the ID of the attribute is *com.ibm.myattribute* you can read the value using

```
workitem.getValue("com.ibm.myattribute")
```

- __d. Add the following code to read the attributes and print the value using the attribute ID's

```
var estimatedInvest = workItem.getValue(
    "com.acme.openup.workitem.attribute.estinvest");
console.log("Estimated Invest: " + estimatedInvest);

var estimatedEffort =workItem.getValue(
    "com.acme.openup.workitem.attribute.esteffort");
console.log("Estimated Effort: " + estimatedEffort);

var averageCost = workItem.getValue (
    "com.acme.openup.workitem.attribute.cost");
console.log("AverageCost: " + averageCost);
```

- __e. Now modify the return statement to calculate the total cost to:

```
var totalCost = new
String(parseInt(estimatedEffort)*parseInt(averageCost)
+parseInt(estimatedInvest));
if(parseInt(totalCost)>0){
    console.log("- Total cost: " + totalCost);
    return totalCost;
}
console.log("- return no cost");
return "";
```

These statements calculate the total cost and return either a number or a blank value. The blank value is necessary for the next lab.

__f. The complete function should now look like below

```
dojo.provide("com.acme.providers.script.TotalCostScriptedCalculatedValue");
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");

(function() {

    dojo.declare("com.acme.providers.script.TotalCostScriptedCalculatedValue", null, {

        getValue: function(attribute, workItem, configuration) {
            console.log("- Start");

            var estimatedInvest = workItem.getValue("com.acme.openup.workitem.attribute.estinvest");
            console.log("Estimated Invest: " + estimatedInvest);
            var estimatedEffort =workItem.getValue("com.acme.openup.workitem.attribute.esteffort");
            console.log("Estimated Effort: " + estimatedEffort);
            var averageCost = workItem.getValue ("com.acme.openup.workitem.attribute.cost");
            console.log("AverageCost: " + averageCost);

            var totalCost = new String(parseInt(estimatedEffort)*parseInt(averageCost)+parseInt(estimatedInvest));
            if(parseInt(totalCost)>0){
                console.log("- Total cost: " + totalCost);
                return totalCost;
            }
            console.log("- return no cost");
            return "";
        }
    });
})();
```

__15. Save the changes to the script.

__a. Go back to the Attribute Customization editor.

If you have closed it or navigated away from it, open the *Project Area* editor, switch to the *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section. Locate the customization type *Calculated Value*, expand its node and click at the customization.

__i. In the Configuration editor section press **Reload** to upload the changes.

__ii. **Save** the Process Configuration change.

If using RTC 4.0.3 or greater



Follow instructions in [Appendix D – Changes in 4.0.3 attribute customization script based editor](#) for guidelines in how to follow the lab using the new editor features.

__b. Try out your script.

- __i. Create a new work item of type Technology Review. Switch to the Tech Review Details tab it should now look like below:

Estimated Effort:	<input type="text"/>
Average Cost:	<input type="text"/>
Estimated Investment:	<input type="text"/>
Total Cost:	<input type="text"/>

Overview	Tech Review Details	Links	Approvals	History	
----------	---------------------	-------	-----------	---------	--

- __ii. Open the log `C:\JSWorkshops\Workspaces\Lab1\.metadata\.log` and scroll way down to the bottom. You should see something similar to

```
!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:34:54.009
!MESSAGE LOG: - Start

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:34:54.011
!MESSAGE LOG: Estimated Invest:

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:34:54.013
!MESSAGE LOG: Estimated Effort:

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:34:54.015
!MESSAGE LOG: AverageCost:

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:34:54.016
!MESSAGE LOG: - return no cost
```

- __c. Enter some plain values such as *11, 100, 3000*. Once values are provided, the total cost is calculated from the values.

Estimated Effort:	<input type="text" value="11"/>
Average Cost:	<input type="text" value="100"/>
Estimated Investment:	<input type="text" value="3000"/>
Total Cost:	<input type="text" value="4100"/>

Overview	Tech Review Details	Links	Approvals	History	
----------	---------------------	-------	-----------	---------	--

- __d. The first version of your script is now working!

- __16. There are several issues with the first attempt.

It is not able to handle input such as 10d for 10 days or 10T for 10 thousand.

You would like to do debugging, but you want to be able to switch debugging off.

- __17. Lets address the debug issue first.

- __a. After the `function` declaration, define a new variable `doDebug` to switch debug on and off and set it to **true**.

```
var doDebug = true;
```

- __b. After that define a new variable for `scriptname` like the line below.

```
var scriptname = "TotalCostScriptedCalculatedValue";
```

- __c. Your script should now look like below

```
(function() {
  var doDebug = true;
  var scriptname = "TotalCostScriptedCalculatedValue";
  dojo.declare("com.acme.providers.script.TotalCostScri
```

- __d. Define a new function called `debug()` that takes a variable and logs if `doDebug` is true.

- __i. Define the function after the `return"";` statement in the body of the function you are developing. The code looks like

```
function debug(display) {
  if(doDebug) {
    console.log(scriptname + " " + display);
  }
}
```

- __e. Replace the console.log calls by calls to the new function debug().

Your implementation should now look like below.

```
(function() {
    var doDebug = true;
    var scriptname = "TotalCostScriptedCalculatedValue";

    dojo.declare("com.acme.providers.script.TotalCostScriptedCalculatedValue", {

        getValue: function(attribute, workItem, configuration) {
            debug("-- Start");

            var estimatedInvest = workItem.getValue("com.acme.openup.workitem.estimatedInvest");
            debug("Estimated Invest: " + estimatedInvest);
            var estimatedEffort = workItem.getValue("com.acme.openup.workitem.estimatedEffort");
            debug("Estimated Effort: " + estimatedEffort);
            var averageCost = workItem.getValue("com.acme.openup.workitem.averageCost");
            debug("AverageCost: " + averageCost);

            var totalCost = new String(parseInt(estimatedEffort)*parseInt(averageCost));
            if(parseInt(totalCost)>0){
                debug("-- Total cost: " + totalCost);
                return totalCost;
            }
            debug("-- return no cost");
            return "";
        }

        function debug(display){
            if(doDebug){
                console.log(scriptname + " " + display);
            }
        }
    });
})
```

- __i. **Save** your changes to the script.
- __f. Go back to the Attribute Customization editor.
- __i. Press **Reload** to upload the changes.
- __ii. **Save** the change to the Process Configuration.
- __g. Test your script.
- __i. Create a new work item of type *Technology Review*.
- __ii. Test if the script is still working.
- __iii. Check the log file for the output.

- __h. You can now turn the logging on and off by setting `doDebug` to **true** or **false**. This is very useful if you have more than one script. You can individually switch the scripts debug output on and off and avoid too much clutter.
- __i. In addition the script name is printed in front of each debug statement which allows highlighting the block of outputs related to one script easier when using advanced text editors. See the example below:

```
!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:50:18.030
!MESSAGE LOG: TotalCostScriptedCalculatedValue - Start

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:50:18.032
!MESSAGE LOG: TotalCostScriptedCalculatedValue Estimated Invest:

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:50:18.034
!MESSAGE LOG: TotalCostScriptedCalculatedValue Estimated Effort:

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:50:18.037
!MESSAGE LOG: TotalCostScriptedCalculatedValue AverageCost:

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-12-10 12:50:18.039
!MESSAGE LOG: TotalCostScriptedCalculatedValue - return no cost
```

- __18. What is left is to convert the values with modifiers such as m, d, M, T, calculate the real value and use the result in the calculation.

- __a. Add the following lines in front of the **var totalCost = ...** statement:

```
estimatedInvest=convertValue(estimatedInvest);
estimatedEffort=convertValue(estimatedEffort);
averageCost=convertValue(averageCost);
```

- __b. Add a new function after the `debug()` function to do the conversion. The code looks like:

```
function convertValue (value){
    debug("convertValue");
    var calc = handleUndefined(value);
    var slen=calc.length;
    var modifier = calc.substring(slen-1, slen);
    var factor = getFactor(modifier);
    if(factor!=""){
        numbervalue=calc.substring(0, slen-1);
        calc = numbervalue*factor;
    }
    return calc;
}
```

The code calls `handleUndefined()` to convert empty input into a string containing "0".

With the now valid assumption that the string is at least of length one, the code gets the

last digit as modifier. The modifier is used in `getFactor()` to calculate the multiplier. If a factor can be calculated, that digit is removed from the input and the true value is calculated. If there is no factor the number is passed through.

- __c. The missing function to handle undefined values needs to be implemented. Insert it after the `convertValue()` function. Add the code below.

```
function handleUndefined (value){
    if(value==null){
        value="0";
    }
    if(value==""){
        value="0";
    }
    return value;
}
```

- __d. The last step that is missing is to implement getting the factor.

- __i. Add a new function `getFactor()` after the last function, then save the JavaScript file. The code would look like below:

```
function getFactor(modifier){
    // m, w, d, h for month, weeks, days...
    // M, T for millions and thousands
    debug("getFactor");
    var factor = "";
    switch (modifier) {
        case 'w':
            factor = "40";
            break;
        case 'm':
            factor = "240";
            break;
        case 'd':
            factor = "8";
            break;
        case 'h':
            factor = "1";
            break;
        case 'T':
            factor = "1000";
            break;
        case 'M':
            factor = "1000000";
            break;
        default:
            factor = "";
            break;
    }
    debug("getFactor:" + factor);
    return factor;
}
```


- __19. Test the completed script.
 - __a. Go back to the *Attribute Customization* editor.
 - __i. Press **Reload** to upload the changes.
 - __ii. **Save** the Process configuration change.
 - __b. Try out your script.
 - __i. Create a new work item of type Technology Review.
 - __ii. Enter values for example 2d, 1T, 1M and check if the calculation is performed correctly.

Estimated Effort:	2d
Average Cost:	1T
Estimated Investment:	1M
Total Cost:	1016000

Overview | Tech Review Details | Links | Approvals | History

- __c. If your script has errors, inspect the log output and fix them.
- __20. If you fix errors, save and reload the script and save the Process Configuration every time before you test the changes. Make sure to create a new work item or at least to close and reopen the editor for an existing work item to make sure the change to the script and the Process Configuration gets picked up by the editor.

You have just successfully completed your first script based calculated value provider.

5.2.2 “AttributeValueAnalyzer” Calculated Value Provider

Section Scenario



You are curious what information you can gather from work item attribute values. You decide to create a small spy tool that can help you better understand how to access data using scripts and what data the script gets.

The script returns data for attributes and their values as a multi line string. It can be used in test environments to make the development of script based attribute value providers easier.

- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user **jim** password **jim**.
 - __a. Open the Project Explorer.
 - __b. Create a Copy of the `TotalCostScriptedCalculatedValueProvider.js` and save it as `AttributeValueAnalyzerCalculatedValueProvider.js`.
 - __c. Open the new script `AttributeValueAnalyzerCalculatedValueProvider.js` in a new editor.
 - __d. Remove the functions you added to the `TotalCostScriptedCalculatedValueProvider` `getFactor()`, `handleUndefined()`, and `convertValue()`, except for the `debug()` function.
 - __e. Delete the code between the line `debug(" - Start");` and the final return statement.
 - __f. Rename all occurrences of the text **TotalCostScriptedCalculatedValue** to **AttributeValueAnalyzer**. You should have 3 replacements:
 - __i. The `dojo.provide()` statement at the beginning should now show:


```
dojo.provide("com.acme.providers.script.AttributeValueAnalyzer");
```
 - __ii. The `dojo.declare()` statement should now show:


```
dojo.declare("com.acme.providers.script.AttributeValueAnalyzer");
```
 - __iii. The `scriptname` statement should now show


```
var scriptname = "AttributeValueAnalyzer";
```

- __g. In the line after the `var scriptname =` statement add this new declaration that gives you access to the built in Attributes:

```
var WorkItemAttributes =
com.ibm.team.workitem.api.common.WorkItemAttributes;
```

This defines a variable that allows easy access to the API defined in the corresponding `dojo.require()` statement as the second line of the script.

- __h. Below the first debug statement in the function declare a new attribute

```
var out = "Attribute Values:\n";
```

The new variable `out` will contain all the data to be returned later.

- __i. Change the return statement to return the value of `out` The line should look like:

```
return out;
```

- __j. The Script should now look like below:

```
* Licensed Materials - Property of IBM
dojo.provide("com.acme.providers.script.AttributeValueAnalyzer");
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");

(function() {
    var doDebug = true;
    var scriptname = "AttributeValueAnalyzer";
    var WorkItemAttributes = com.ibm.team.workitem.api.common.WorkItemAttributes;

    dojo.declare("com.acme.providers.script.AttributeValueAnalyzer", null, {

        getValue: function(attributeId, workItem, configuration) {
            debug("- Start");

            var out = "Attribute Values:\n";

            return out;

            function debug(display) {
                if(doDebug) {
                    console.log(scriptname + " " + display);
                }
            }
        }
    });
})();
```

__2. Now add a new function that handles analyzing the general information from the work item attribute.

__a. Add the code below after the debug function code

```
function getAttributeData (message,attributeID){
  debug("Get Attribute Data for " + attributeID);
  var attributeValue="";
  var attributeLabel="";
  var result = message + "\n";
  try{
    isSet=workItem.isAttributeSet(attributeID);
    attributeValue=workItem.getValue(attributeID);
  } catch (e) {
    attributeValue = "Exception reading the attribute value";
  }
  try{
    attributeLabel=workItem.getLabel(attributeID);
  } catch (e) {
    attributeLabel= "Exception reading the attribute label";
  }
  result += "Set: " + isSet + "\nValue= " + attributeValue +
    "\nLabel= " + attributeLabel;
  debug(result);
  return result + "\n\n";
}
```

__b. The code tries to access an attribute given an attribute ID. It tries to read the attribute value and the attribute label. The data is sent to the debug console and returned as a readable string.

- __3. To use the code, add statements like the following in front of the return statement of the main function:

```
out+=getAttributeData(<someText>, <WorkItemAttributes.ID>);
```

The lines below show some examples. You can copy all lines from below into your script, Add at least the lines for the *ID*, the *summary*, the *priority*, the *owner* and the *creation date*.

```
out+=getAttributeData("ID: ", WorkItemAttributes.ID);
out+=getAttributeData("TYPE: ", WorkItemAttributes.TYPE);
out+=getAttributeData("Summary: ", WorkItemAttributes.SUMMARY);
out+=getAttributeData("Creator: ", WorkItemAttributes.CREATOR);
out+=getAttributeData("Owner: ", WorkItemAttributes.OWNER);
out+=getAttributeData("Project Area: ",
    WorkItemAttributes.PROJECT_AREA);
out+=getAttributeData("State: ", WorkItemAttributes.STATE);
out+=getAttributeData("Resolution: ",
    WorkItemAttributes.RESOLUTION);
out+=getAttributeData("Found in: ", WorkItemAttributes.FOUND_IN);
out+=getAttributeData("Severity: ", WorkItemAttributes.SEVERITY);
out+=getAttributeData("Priority: ", WorkItemAttributes.PRIORITY);
out+=getAttributeData("Creation Date: ",
    WorkItemAttributes.CREATION_DATE);
out+=getAttributeData("Due Date: ", WorkItemAttributes.DUE_DATE);
out+=getAttributeData("Estimate: ", WorkItemAttributes.ESTIMATE);
out+=getAttributeData("Corrected Estimate: ",
    WorkItemAttributes.CORRECTED_ESTIMATE);
out+=getAttributeData("Time Spent: ",
    WorkItemAttributes.TIME_SPENT);
out+=getAttributeData("Filed Against: ",
    WorkItemAttributes.FILED_AGAINST);
out+=getAttributeData("Planned For: ",
    WorkItemAttributes.PLANNED_FOR);
out+=getAttributeData("Resolution Date: ",
    WorkItemAttributes.RESOLUTION_DATE);
out+=getAttributeData("Tags: ", WorkItemAttributes.TAGS);
out+=getAttributeData("Modified Date: ",
    WorkItemAttributes.MODIFIED);
out+=getAttributeData("Modified By: ",
    WorkItemAttributes.MODIFIED_BY);
```

- __a. Check the script for any errors and save the script.

- __4. You now need to configure the script.

Open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.

- __a. In the Attribute Customization editor, select Calculated Values. Use the **Add...** button at the bottom of the editor to add a new Calculated Value.

- __i. Name the configuration **AttributeValueAnalyzer**.

- __ii. Make sure the Configuration Type is *Calculated Values*.
 - __iii. Select *Script Based Calculated Value* as Provider
 - __iv. Use the **OK** button to create the customization.
- __b. Now upload the script.
- __i. Use the browse button and browse for the script in the *C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization*
 - __ii. Select the file *AttributeValueAnalyzerCalculatedValueProvider.js*.
- __c. Save the change to the Process Configuration.
- __5. Configure the calculated value provider
- __a. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
 - __i. Select any work item type and edit the *Description* attribute.
 - __ii. Select the *AttributeValueAnalyzer* for the calculated value provider.
 - __iii. Add dependencies to the attributes you are interested in triggering the provider.

You can basically select all attributes except the *Modified Date*. Only changes to the selected attributes will trigger an update. If you added only selected lines above, add dependencies to the attributes you monitor. At least add *ID*, the *summary*, the *priority*, the *owner* and the *creation date*.
 - __b. Click **OK** and **save** the changes to the process configuration.
- __6. Now you can test the script.

How this Calculated Value works

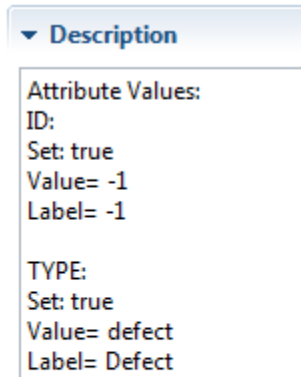


You have now created a JavaScript based calculated value for the **Description** attribute that will calculate the content of the description from the work item attribute values. This is done every time an attribute that is in the list of dependent attributes changes. Since this description attribute is very prominent in the UI, it makes looking at the attribute data real easy.

The only disadvantage is, that this is only usable in a script test environment, because it would overwrite all descriptions for work items.

- __a. Create a new work item.

- __i. Look at the **Description Attribute** in the work item editor. The work item Description attribute should now show the calculated text with the information about the attribute values like in the screen shot below.

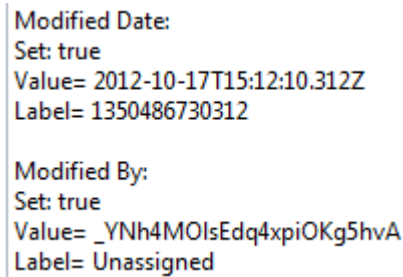


- __ii. Save the work item. After saving, the type and the state of the work item are set as well as creation and modification time. The ScriptBased calculated value is triggered by the changes and the Description attribute should show more data.
- __iii. Scroll down and look at the data. Especially look at information such Enumerations like priority, Creator, Owner, Filed Against and other item type attributes.

You can see the Enumeration literals as well as the enumeration labels.

Item type attributes show an ID String for the item and the label displays user and other element names.

- __a. Look at the Modified Date output:



It is somewhat unreadable. To show the data correctly some conversion is necessary.

- __7. Extend your script to show human readable dates that could also used for calculation.

- __a. At the beginning of the script underneath the first `dojo.require()` statement add the following new `dojo.require()` statements.

```
dojo.require("dojo.date");
dojo.require("dojo.date.stamp");
```

This API can be used to access and convert the timestamp type attributes.

- __b. Modify the code and add the boldface lines below each date attribute you have in your script. If you have not added the dates to begin with, add all the code from below.

- __i. The Creation Date:

```
out+=getAttributeData("Creation Date: ",
WorkItemAttributes.CREATION_DATE);
var creationDate=
dojo.date.stamp.fromISOString(workItem.getValue(WorkItemAttr
ibutes.CREATION_DATE));
out+= "Creation Date " + creationDate + "\n\n";
```

- __ii. The Due Date:

```
out+=getAttributeData("Due Date: ",
WorkItemAttributes.DUE_DATE);
var dueDate=
dojo.date.stamp.fromISOString(workItem.getValue(WorkItemAttr
ibutes.DUE_DATE));
out+= "Due Date " + dueDate + "\n\n";
```

- __iii. The Modified Date

```
out+=getAttributeData("Modified Date: ",
WorkItemAttributes.MODIFIED);
var modifiedDate=
dojo.date.stamp.fromISOString(workItem.getValue(WorkItemAttr
ibutes.MODIFIED));
out+= "Modified Date " + modifiedDate + "\n\n";
```

This code allows to show the date in a human readable format.

- __c. **Save** the script.

- __8. Reload the script.

- __a. Navigate to the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __b. Open the calculated values node on the attribute customization editor and click on the *AttributeValueAnalyzer* calculated value provider.

- __c. On the Configuration editor press the **Reload** button.
 - __d. **Save** the Process Configuration changes.
- __9. Test the changes to the script.
- __a. Create a work item. Enter the summary and the category then save the work item.
 - __b. Validate that the date attributes now return a readable date.

```
Creation Date:
Set: true
Value= 2012-10-29T15:39:15.033Z
Label= 1351525155033
```

```
Creation Date Mon Oct 29 2012 16:39:15 GMT+0100 (CET)
```

- __10. Finally add a section to show some of your custom attributes.
- __a. You can use the function `getAttributeData()` with the custom attributes by providing the custom attribute ID's. Place the cursor behind the line

```
out+=getAttributeData("Modified By: ",
WorkItemAttributes.MODIFIED_BY);
```

- __b. Add the following lines

```
out+="\nCustom:\n";
out+=getAttributeData("Estimated Invest: ",
"com.acme.openup.workitem.attribute.estinvest");
out+=getAttributeData("Estimated Effort: ",
"com.acme.openup.workitem.attribute.esteffort");
out+=getAttributeData("Average Cost: ",
"com.acme.openup.workitem.attribute.cost");
out+=getAttributeData("Total Cost: ",
"com.acme.openup.workitem.attribute.totalcost");
```

Process IDs in your scripts



Note the process IDs that are used in the scripts as in the snippet above. You will have to adjust the IDs to the naming convention used if you followed a different.

- __c. **Save** the script.
- __11. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.

- __a. Select the work item type **Technology Review** and edit the *Description* attribute. If you select another work item type, you won't see the attributes you want to add as dependencies.
 - __b. Add dependencies to the custom attributes **Cost**, **Estimated Effort**, **Estimated Investment** and **Total Cost** to trigger the calculation on changes.
 - __c. Follow the steps from step 8 on page 151 to reload the script and save the process configuration.
- __12. Test the changes
- __a. **Create a new** work item of type **Technology Review** and verify the *Tech Review Details* tab shows up.
 - __b. Change the values of the custom attributes **Cost**, **Estimated Effort**, **Estimated Investment** on the *Tech Review Details* tab and make sure they are presented in the description of the work item.
- __13. Remove the calculated value for the description attribute.
- __a. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
 - __b. Select any work item type and edit the **Description** attribute.
 - __c. Select *None* for the calculated value provider.
 - __d. Remove the dependencies you added to the description attribute.
 - __e. Click **OK** and **save** the changes to the process configuration.

5.2.3 Summary

You have successfully implemented a calculated value provide that can be used in test environments to understand the values of work item attributes in script based customization.

5.3 Script Based Conditions

Section Scenario

Your project wants to enforce the following:

The Total Cost estimate is value is calculated in the work item if all the following are true:

- The state is “Experimental”
- The complexity is low or manageable
- The impact is High.



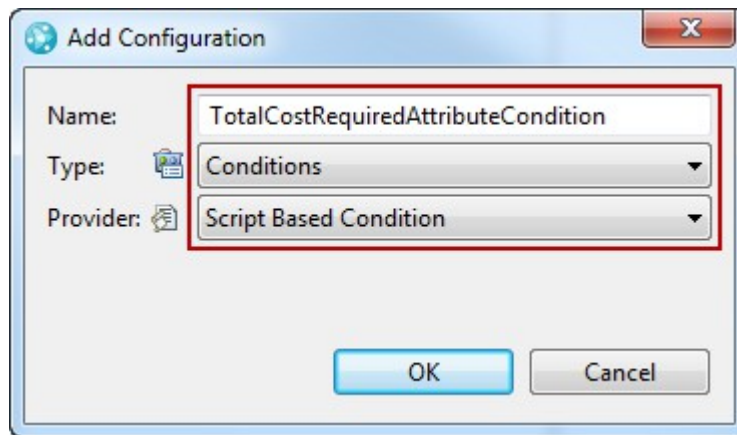
Since you manage to exceed expectations so far, you are again asked to provide a solution. Your experience working with RTC conditions has already helped you figure out that a simple condition won't support your needs. Therefore you decide to look into script based conditions.

You will write a condition that checks the attribute values one by one to decide if the work item is in the desired condition.

Rational Team Concert supports complex required attributes scenarios using a combination of **operational behavior** – a precondition – and conditions based on JavaScript.

- __1. Open the Eclipse client if it is not already opened and connect to your repository. Use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user **jim** password **jim**.
- __2. The first step is to create a condition to calculate if an attribute is required. You will now implement this condition.
 - __a. Open the project area editor for *Nifty Application Project* project area, switch to the *Process Configuration* tab and navigate to **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __b. In the Attribute Customization editor, select Conditions. Use the **Add...** button at the bottom of the editor or use a right click to add a new Condition.

- __c. Name the condition **TotalCostRequiredAttributeCondition** make sure the type shows Conditions and select the only option Script Based Condition for the provider. Then click **OK** to create it.



- __d. To download the example code, click at the **Download example** link in the Configuration editor.

- __i. Browse to the folder
`C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\`
- __ii. Change the file name to ***TotalCostRequiredAttributeCondition.js***
- __iii. Press the **Save** button.

- __e. **Save** the process configuration.

- __3. To see the new script in Eclipse open the *Project Explorer* view.

- __a. Select the *ScriptBasedCustomization* project
- __b. Use the *Refresh* context menu entry or press *F5*.
- __c. The externally saved script file now shows up in Eclipse
- __d. Open the file for editing and review it.

- __i. The file should look like in the image below

```

+ * Licensed Materials - Property of IBM
dojo.provide("com.example.Condition");

(function() {
  dojo.declare("com.example.Condition", null, {
    matches: function(workItem, configuration) {
      return true;
    }
  });
})();

```

How the Script Based Condition works



The condition implements the *matches* function. The *boolean* result of this function is what the precondition will use to determine if the expected condition is met and to consequently react.

- __4. Start implementing your own code.

__a. To provide a useful name change the ID *"com.example.Condition"* in the `dojo.provide` and `dojo.declare` statement to *"com.acme.providers.script.TotalCostRequiredAttributeCondition"*

__b. Prepare for debugging similar to the other scripts.

- __i. Add the following statements after the first *function()* statement.

```

var doDebug = true;
var scriptname = "TotalCostRequiredAttributeCondition";

```

__ii. Copy the function **debug()** from the calculated value provider script `TotalCostScriptedCalculatedValue.js` and insert it after the last return statement. You are now able to debug your work.

__iii. Add this line before the return statement. It will tell you if the condition ran:

```

debug("- Start");

```

- __5. The basic preparation work is now finished. Now the script needs to be implemented to match the requirement.

__6. At first it is interesting to know the state of the work item. See the [Attribute customization Wiki entry](#) for more information.

__a. Underneath the `dojo.provide` statement, add a line

```
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");
```

As mentioned, this line is required to have access to the `WorkItemsAttributes` object that provides with the constants that will ease the access to the built in attributes.

__b. Add the work item object behind the function declaration using the lines below.

```
var WorkItemAttributes=
com.ibm.team.workitem.api.common.WorkItemAttributes;
```

The following image shows the added code.

```
* Licensed Materials - Property of IBM
dojo.provide("com.acme.providers.script.TotalCostRequiredAttributeCondition");
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");

(function() {
    var doDebug = true;
    var scriptname = "TotalCostRequiredAttributeCondition";
    var WorkItemAttributes = com.ibm.team.workitem.api.common.WorkItemAttributes;

    dojo.declare("com.acme.providers.script.TotalCostRequiredAttributeCondition", null, {
        matches: function(workItem, configuration) {
            debug("Start");
        }
    });
});
```

__7. Check the interesting attributes of the work item to determine if the condition has to check for the other values. The first attribute to check is the state of the work item.

__a. Find the state ID's of the work item, in order to check when the condition has to check for the other values. There are basically two options to find them.

__i. Option 1: You can look up the workflow name of the workflow used by the work items you are interested in.

__ii. Option 2 is just to use a script to get the value, either using the `AttributeValueAnalyzerCalculatedValueProvider.js` or to debug the script you develop and use the debug output to find out the state ID's.

__b. You have used the second option already so this time you look up the information in the *Process Configuration*.

__i. The information can be found in the in the Eclipse client in **Process Configuration > Project Configuration > Configuration Data > Work Items > Workflows**.

- __ii. Browse the workflows and find the one you are interested in. In this case the Workflow is named “*Technology Review Workflow*”. Copy the name of the workflow.
- __c. To find the state ID's, open the *Process Configuration Source* tab and use *CTRL+F* to open the Eclipse Find window. Paste or type the workflow name “*Technology Review Workflow*” into the Find field of the *Find and Replace* dialog and press the *Find* button.
 - __i. Find the *State* entries and look for the state name “*Experimental*” and look up the id property.

```
<action id="tecReviewWorkflow.action.a8" name="Propose" state="tecReviewWorkflow.state.s1"/>
<state group="open" id="tecReviewWorkflow.state.s1" name="Proposed" showResolution="false">
  <action id="tecReviewWorkflow.action.a1"/>
</state>
<state group="inprogress" id="tecReviewWorkflow.state.s2" name="Under Research" showResolution="false">
  <action id="tecReviewWorkflow.action.a2"/>
  <action id="tecReviewWorkflow.action.a3"/>
  <action id="tecReviewWorkflow.action.a4"/>
</state>
<state group="inprogress" id="tecReviewWorkflow.state.s3" name="Experimental" showResolution="false">
  <action id="tecReviewWorkflow.action.a5"/>
  <action id="tecReviewWorkflow.action.a4"/>
</state>
<state group="inprogress" id="tecReviewWorkflow.state.s4" name="Approved" showResolution="false">
  <action id="tecReviewWorkflow.action.a6"/>
</state>
```

- __d. The Experimental state has the ID "*tecReviewWorkflow.state.s3*"
- __e. The script you are writing is supposed to be valid for the Experimental state. Go back to the *TotalCostRequiredAttributeCondition* script code, and add the lines below in your script in before the final return statement.

```
// Access to the built in work item attribute ID's
var state = workItem.getValue(WorkItemAttributes.STATE);
debug("state: " + state);
if(state!="tecReviewWorkflow.state.s3"){
  return false; // Nothing to do
}
```

The code checks if the state is valid. If not, the script returns false, because in our case there is nothing more to do. The format of the condition has been deliberately chosen. It allows you to easily add more states using `||state="<stateID>"`

- __f. **Save** your script.

Finding Information in the Process Configuration Source

It is possible to look some information up in the process configuration UI. If the location of the information is not obvious, here is a strategy for finding it:



- Find general information such as names in the process configuration UI.
- Open the Process configuration Source tab.
- Use CTRL+F to open the Eclipse Find window.
- Paste or type a part of the name or ID, for example “risk.impact” into the find section of the Find and Replace dialog and press the Find button.
- Look at the xml entries for id's, literal id's and the associate name.
- For your enterprise custom process elements, use a common naming schema for ID's such as com.company.workitem.attribute.attributename which makes searching much easier

__8. Now you want to check if the impact of the Technology Review is **High**.

__a. The first information you need is the attribute ID of the Impact attribute. To gather this information, in the Eclipse client open the project area editor and in the Process Configuration tab navigate to **Project Configuration > Configuration Data > Types and Attributes**.


__i. In the Work Item Type section, select the work item type *Technology Review* and scroll to the *Attributes* section below. Browse for the attribute *Impact* and use the **Edit** button to open the editor for the attribute.

__ii. The Attribute Editor shows the attribute ID. Copy the work item ID to be used later. The value should be *com.ibm.team.workitem.workItemType.risk.impact*.

- __b. To be able to compare the data in the attribute with the possible values it can have, there are two choices: use the enumeration literals or use the label of the enumeration values.

You will use the labels in this example.

Using the literals or the labels in your script



Using the literals makes the coding and debugging harder as you have to deal with the internal IDs assigned to the enumeration values.

On the other hand, if using the labels you can use the human readable representation of the enumeration values, but you'll have to review your scripts if you use the same process and logic in project areas in different locales or if these labels values are changed.

- __c. Identify the label for the Impact enumeration. The one we're interested in is **High**.

- __i. Add the following lines to your script before the final return statement. The code uses the attribute ID you just looked up to get the label of the impact attribute.

```
// Check for the impact
var impact = workItem.getLabel(
    "com.ibm.team.workitem.workItemType.risk.impact");
debug("impact: " + impact);
if(!(impact=="High")){
    return false;
}
```

The code here simply checks if the impact is high and just exits the condition with false if not. This means the condition will not be valid. The same pattern is used to determine if the condition can exit with false is used in all the code here.

- __9. At last you want to check the value of the Complexity attribute. You will use the label again. for other approaches, use the same strategy as described in step 7 on page 157 above to find the required data.

- __a. The attribute ID is *com.acme.openup.workitem.attribute.complexity*.
- __b. The enumeration labels values to look for are *Low* and *Manageable*.

- __c. Add the following lines to your script before the final return statement.

```
// Check for the complexity
var complexity = workItem.getLabel(
    "com.acme.openup.workitem.attribute.complexity");
debug("complexity: " + complexity);
if(!(complexity=="Low"||complexity=="Manageable")){
    return false;
}
```

The statements check if the work item has a complexity set to qualify it to require the Total Cost estimation. If not, the script returns false.

- __10. At this point in the script the work item is qualified and needs the total cost attribute

- __a. To make the script a little bit prettier, add a comment and a debug statement in front of the final return statement. Check and make sure the final return statement returns true like below.

```
// Total cost attribute is required to have a value.
debug("Total Cost Attribute Required");
return true;
```

- __11. **Save** the script

- __12. The script is now ready for testing and all that is left is to upload and test it.

The Condition is Not Assigned to the Attribute



Unlike other attribute customization you've done so far, Conditions are not configured in the attribute section. Conditions are activated through operation behavior.

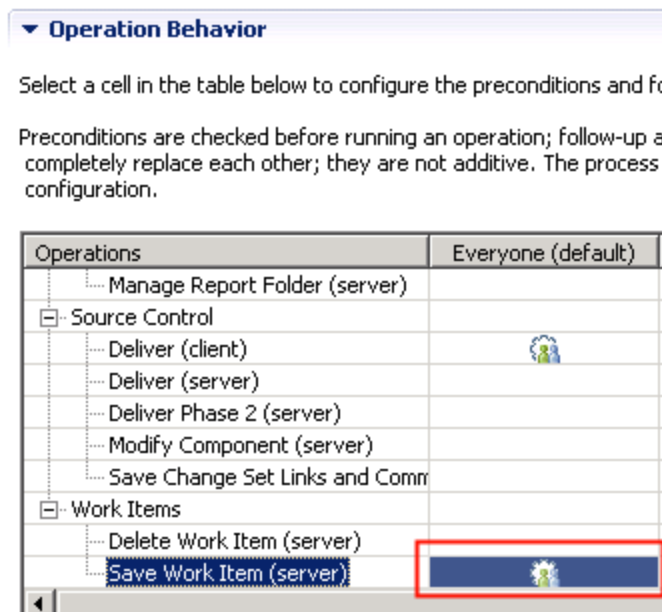
- __a. To upload the script open the Process Configuration tab in the project area editor and go to **Project Configuration > Configuration Data > Work Items > Attribute Customization**
- __b. Expand the list of conditions and select the condition *TotalCostRequiredAttributeCondition* you created for the lab
- __i. In the Configuration editor select the browse button to browse for the script file.
- __ii. Browse to the folder *C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization*, select the file ***TotalCostRequiredAttributeCondition.js*** and press **Open**.

- __iii. Verify that the *Attachment Path* changed to
/workitem/scripts/common/TotalCostRequiredAttributeCondition.js
and the *Class Name* changed to
com.acme.providers.script.TotalCostRequiredAttributeCondition
- __iv. **Save** the changes in the Attribute Customization wizard.

__13. Now the script should be ready for testing. You need to configure the operational behavior before you can test if it works.

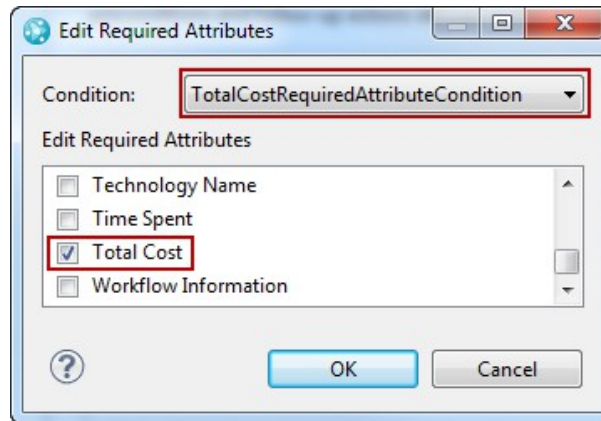
__a. In the Process Configuration tab of the project area editor, browse to **Configuration > Team Configuration > Operation Behavior**.

__i. Search for the **Save Work Item (server)** operation for the role *Everyone* in the related table column. It should be at the end of the operations list. Click at the operation.



- __ii. In the editor below click the **Add** button to add a precondition and browse the available choices.
- __iii. Find the **Required Attributes For Condition** precondition which is an option that could be used in this case.
- __iv. Click **OK** to add the precondition.

- __v. With the just added precondition highlighted, a configuration section for it appears in the right side of the editor window. Select the **Add** button in the right editor window.
- __a. Select the **TotalCostRequiredAttributeCondition** as *Condition* in the Required Attributes editor.
- __b. Browse the attributes and select the **TotalCost** attribute to be required based on the condition.



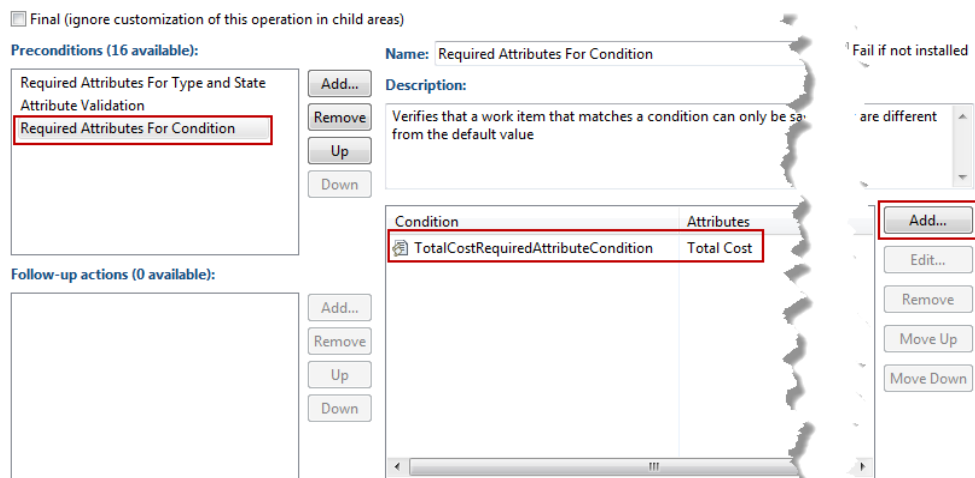
- __c. Click **OK**

Configure a Condition for an attribute




The in this step you defined which attribute is required in case the condition returns true. You can use one condition for several attributes.

- __vi. Your editor should look like below.



- __b. **Save** the changes to the process configuration.

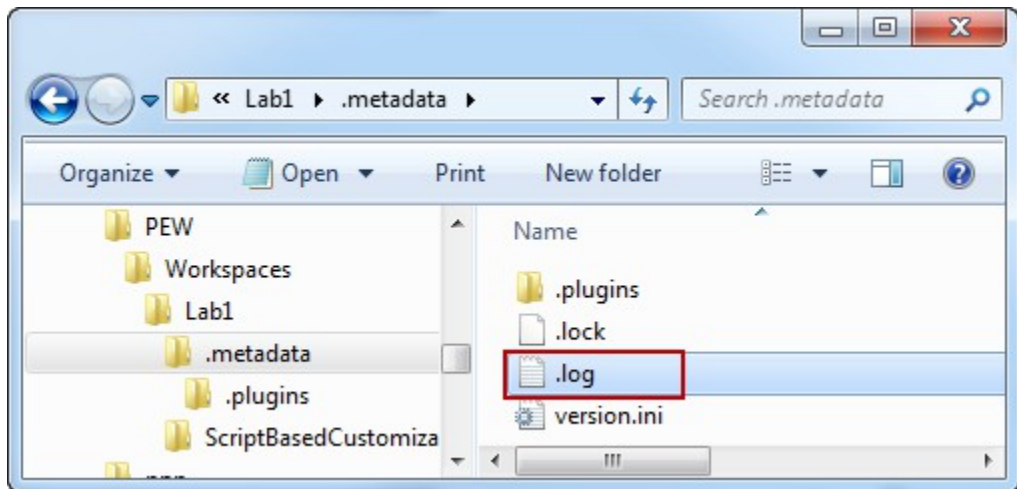
Conditions for Dynamic Read-only Attributes



The operational behavior **Read Only Attributes For Condition** can be used to dynamically switch work items attributes to read only.

- __14. Now the condition can be tested.

- __a. It is important to monitor the test in the log files. See 13 at page 133 for more details about the logging capability.
- __b. Use the Eclipse client to create a new work item of type Technology Review.
- __c. Script logging:
 - __i. Open an explorer window, locate the Eclipse log file `C:\JSWorkshops\Workspaces\Lab1\.metadata\.log` and open it with a text editor (editors like [Notepad++](#) work best).



- __ii. Scroll to the end of the log file. It should show a log entry of the script engine and the log message like below.

```
!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:23:19.495
!MESSAGE LOG: TotalCostRequiredAttributeCondition Start

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:23:19.498
!MESSAGE LOG: TotalCostRequiredAttributeCondition state:
```

At this point, the new work item has no state.

- ___d. In the work item provide a *Summary* such as “*Testing TotalCost condition*” and **save** the work item.
- ___e. Use the “Research” action to change the state of the work item to “Under Research” and **save** the change.
- ___f. Use the “Test” action to change the state of the work item to “Experimental” and **save** the change.
- ___g. Change to the **Tech Review Details** tab.
- ___h. Change the *Complexity* to **Manageable** or **Low**.
- ___i. Change the *Impact* to **High**.
- ___j. You should see a red asterisk popping up near the *Total Cost* attribute presentation.

Estimated Effort:	<input type="text"/>		
Average Cost:	<input type="text"/>		
Estimated Investment:	<input type="text"/>	Complexity:	<input type="text" value="Manageable"/>
Total Cost*:	<input type="text"/>	Impact:	<input type="text" value="High"/>

- ___k. Reload the log file. You should now see an output similar to below.

```
!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:29:59.976
!MESSAGE LOG: TotalCostRequiredAttributeCondition Start

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:29:59.978
!MESSAGE LOG: TotalCostRequiredAttributeCondition state: tecReviewWorkflow.state.s3

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:29:59.980
!MESSAGE LOG: TotalCostRequiredAttributeCondition impact: High

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:29:59.982
!MESSAGE LOG: TotalCostRequiredAttributeCondition complexity: Manageable

!ENTRY com.ibm.team.rtc.common.scriptengine 1 0 2012-11-08 16:29:59.984
!MESSAGE LOG: TotalCostRequiredAttributeCondition Total Cost Attribute Required
```

- ___l. If not, review the script and fix any issues. The final script is presented below

___15. Test another scenario.

- ___a. Create a work item of type defect.
- ___b. Modify the work item a little bit and reload the log file.
- ___c. You should see – nothing. The precondition triggers the script only for work items that have the configured attribute in the precondition (*Total Cost*), and the defect does not have it.

__16. As reference, the final script code should look as follows.

```

* Licensed Materials - Property of IBM
dojo.provide("com.acme.providers.script.TotalCostRequiredAttributeCondition");
dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");

(function() {
    var doDebug = true;
    var scriptname = "TotalCostRequiredAttributeCondition";
    var WorkItemAttributes = com.ibm.team.workitem.api.common.WorkItemAttributes;

    dojo.declare("com.acme.providers.script.TotalCostRequiredAttributeCondition", null, {

        matches: function(workItem, configuration) {
            debug("Start");

            // Access to the built in work item attribute ID's
            var state = workItem.getValue(WorkItemAttributes.STATE);
            debug("state: " + state);
            if(state!="tecReviewWorkflow.state.s3"){
                return false; // Nothing to do
            }
            // Check for the impact
            var impact = workItem.getLabel("com.ibm.team.workitem.workItemType.risk.impact");
            debug("impact: " + impact);
            if(!(impact=="High")){
                return false;
            }
            // Check for the complexity
            var complexity = workItem.getLabel("com.acme.openup.workitem.attribute.complexity");
            debug("complexity: " + complexity);
            if(!(complexity=="Low"||complexity=="Manageable")){
                return false;
            }
            // Total cost attribute is required to have a value.
            debug("Total Cost Attribute Required");
            return true;

            function debug(display){
                if(doDebug){
                    console.log(scriptname + " " + display);
                }
            }
        }
    });
})();

```

You have successfully created and deployed a script based condition for Rational Team Concert that controls when attributes are required.

You can use the same type of conditions together with the **Read-Only Attributes for Condition** operational behavior which allows you to declare attributes read only based on a condition. One example that can be easily derived from this section would be a condition that makes the cost attributes read-only in the states **Approved** and **Adopted** if the impact is high and the complexity low or manageable.

5.4 Script based Validations

Section Scenario



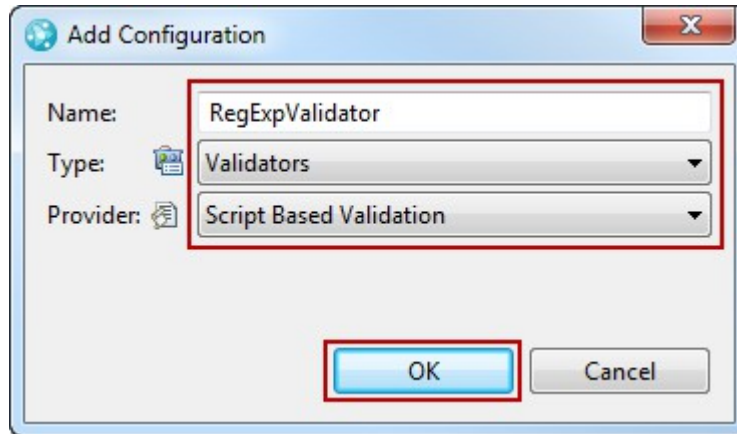
Attribute IDs and other data are hard coded into the script based customization you have done so far. It is impossible to configure them and the scripts can't be reused easily.

Your project is nervous about the cost to maintain all the scripts and potential issues with making sure that the same code in different scripts might be error prone and become unmanageable over time. You have been asked to look into providing a small example that shows that this can be done better.

You will create and debug your own configurable script based validator for regular expressions.

- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user **jim** password **jim**.
- __2. Open the project area editor for the *Nifty Applicaiton Project* project are and switch to the Process Configuration tab.
- __3. The script based validators work the same way the non-script based validators work. This means that the validation can only prevent saving a work items if the Attribute Validation operational behavior precondition is activated, and the validation returns an error. So you need to check for the precondition.
 - __a. In the Process Configuration tab, browse to **Configuration > Team Configuration > Operation Behavior**.
 - __b. Search for the **Save Work Item (server)** operation for the role *Everyone* in the related table column. It should be at the end of the operations list. Click on the operation.
 - __c. Try to find the **Attribute Validation** precondition in the list of preconditions displayed. If the precondition is not configured, use the **Add...** button to add it. Browse for it in the list, select it and select **OK**. The Precondition should now show up in the list of active preconditions.
- __4. You will now implement a script based validation that work with the precondition.
 - __a. Open the **Configuration>Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __b. In the Attribute Customization editor, select *Validators*. Use the **Add...** button at the bottom of the editor or use a right click to add a new Validator.

- __c. Name the new validator **RegExpValidator**, make sure the type shows Validators and select the option Script Based Validation for the provider.



- __d. To download the example code, click at the **Download example** link in the Configuration editor.

- __i. Browse to the folder
`C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\`
- __ii. Change the file name to `RegExpScriptedValidator.js`
- __iii. Press the **Save** button.
- __iv. **Save** the process configuration.

- __5. To see the new script in Eclipse open the *Project Explorer* view.

- __a. Select the *ScriptBasedCustomization* project
- __b. Use the *Refresh* context menu entry or press **F5**.
- __c. The externally saved script file now shows up in Eclipse
- __d. Open the file for editing.

- __6. You will now create the code for the script based validation.
- __a. The initial code looks like below. Review the code. The boxes and the arrows show where you will change the code in the next steps.

```

* Licensed Materials - Property of IBM
dojo.provide("com.example.Validator");

dojo.require("com.ibm.team.workitem.api.common.Severity");
dojo.require("com.ibm.team.workitem.api.common.Status");

(function() {
    ←
    var Severity= com.ibm.team.workitem.api.common.Severity;
    var Status= com.ibm.team.workitem.api.common.Status;

    dojo.declare("com.example.Validator", null, {
        ←
        validate: function(attribute, workItem, configuration) {
            return Status.OK_STATUS;
        }
    });
}());

```

Please review the variable declarations for the variables Status and Severity, and the corresponding dojo.require() statements. This code allows you to create a status with a defined severity.

- __b. Locate the strings "com.example.Validator" and replace it with "com.acme.providers.script.RegExpScriptedValidation" to rename your validator.
- __c. Prepare the code for better debugging as before. In the line after the (function() declaration add the lines (marked by the first arrow):
- ```

var doDebug= true;
var scriptname = "RegExpScriptedValidation";

```
- \_\_d. Open one of the scripts you created before and copy the debug function over. Insert the code below right after the statement return Status.OK\_STATUS (marked by the second arrow).

```

function debug(display) {
 if(doDebug) {
 console.log(scriptname + " " + display);
 }
}

```

- \_\_e. Add a statement to debug directly underneath the validate: function(...) statement.

```
debug("- start");
```

- \_\_7. Now the script is prepared to allow developing the required capabilities. This time you want to keep the script as configurable as possible.

- \_\_a. Review the parameters that the function call retrieves.

```
validate: function(attribute, workItem, configuration) {
```

- \_\_i. The parameter **attribute** provides you with the **ID** of the attribute the provider is configured for.

- \_\_ii. The parameter **workItem** provides access to the work item attribute values.

- \_\_iii. The parameter **configuration** provides access to additional information that can be stored in the process configuration source.

- \_\_b. The JavaScript regular expression class RegExp that you will use requires one parameter with the regular expression string and has an optional parameter for modifiers (such as ignore case). You want the regular expression and the modifier to be parameters that are passed to the validation from outside. Add the following lines to your script right after the line with the debug statement you just added.

```
var
pattern=configuration.getChild("parameters").getStringDefault("pa
ttern", "");
var
modifier=configuration.getChild("parameters").getStringDefault("m
odifier", "");
var verifyPattern=new RegExp(pattern,modifier);
debug("RegExp: "+pattern+" [" +modifier+"]");
```

The code above retrieves information from XML code of the following form that can be added to the process configuration source:

```
<parameters pattern="<PatternString>" modifier="<ModifierString>"
... />
```

- \_\_i. The first line tries to get a <parameters> element inside the configuration information for the validator and tries to read the pattern attribute if available or uses the default empty string. The default is passed as the second parameter to the getStringDefault method.

- \_\_ii. The second line tries the same with an attribute named modifier.

- \_\_iii. The last line prints the result for debugging into the log files.

### Missing Configuration Data Will Throw Exceptions!

If the configuration data in the XML is missing from the process configuration, the call to `configuration.getChild` will retrieve a null value and the subsequent call to `getStringDefault()` will cause an exception. To enhance debugging you can surround these statements with a try /catch block as presented below:



```
try {
 ...
} catch (e) {
 debug("Exception " + e.message);
 throw "Configuration Exception: "
+ e.message;
}
```

- \_\_c. Now it is necessary to retrieve the value of the attribute and test that against the regular expression. Add the following lines to your script, after the debug statement of the last addition.

```
var valuetoTest=workItem.getValue(attribute);
var verifyPattern=new RegExp(pattern,modifier);
var result=verifyPattern.exec(valuetoTest);
debug("Test Result: " + result);
```

- \_\_i. The first line reads the attribute the validator is configured for. The ID of the attribute is passed in the parameter *attribute*.
- \_\_ii. In the next line you create a new regular expression object with the pattern and modifier retrieved from the configuration.
- \_\_iii. The third line is used to execute the regular expression. The regular expression execution returns an array with the [first value containing the pattern match](#).
- \_\_iv. The debug statement just logs the returned values.

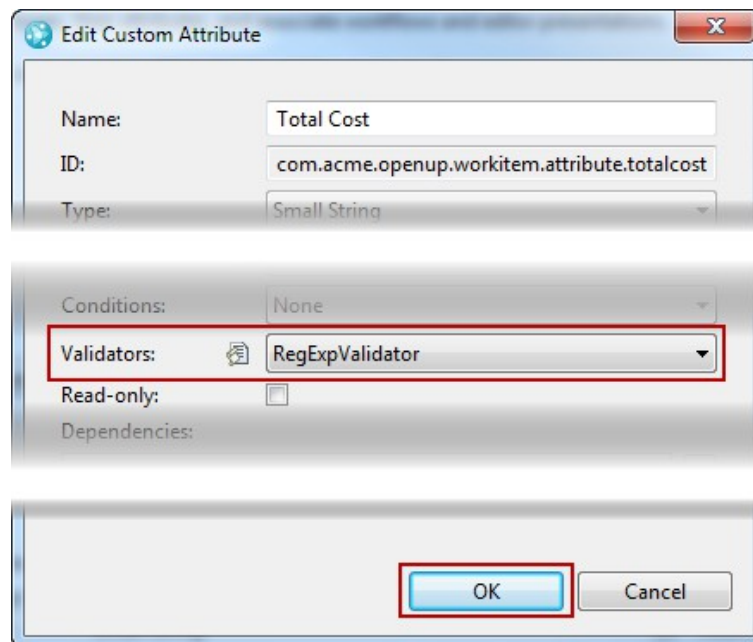
- \_\_d. Now it is possible to test if the value of the attribute matches the pattern. Add the following lines to your script after the last debug statement.

```
if(result[0]!=valuetoTest){
 debug("Validation Fail");
 //Get the configuration parameters about the severity and error
message
 var severity=
configuration.getChild("parameters").getStringDefault("severity",
Severity.ERROR.name);
 var message=
configuration.getChild("parameters").getStringDefault("message",
"Regular Expression failed!");
 debug("Validation: "+severity+" [" +message+"]");
 return new Status(Severity[severity], message);
}
```

- \_\_i. The code tests if the returned result in the array at position 0 equals the whole of the work items attribute we test. The other results in the array contain partial matches of the regular expression. Only if the value of result[0] equals the tested attribute value content used as input value the whole input conforms the expression. If it is the case there was a match. If not, the validation fails, which is handled in the **if** statement.
- \_\_a. First the code logs the failure.
- \_\_b. Then the desired severity is retrieved from the configuration object. As default the severity is set to error.
- \_\_c. Next the message string is retrieved from the configuration. A simple default message is used in case there is no configuration.
- \_\_d. The retrieved information about severity and message is logged.
- \_\_e. Finally a new status object based on severity and message is created and returned.
- \_\_e. Now add a last line to log the succeeded validation after the closing bracket for the if statement:
- ```
debug("Validation Success");
```
- __f. **Save** the changes to your script.
- __g. Reload the modified script.
- __i. Return to the project area editor, the Process Configuration tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __ii. In the Attribute Customization editor, browse for the *RegExpValidator*.

- __iii. Highlight the *RegExpValidator* and click **Reload**. Make sure the Class Name and path now show names containing *RegExpScriptedValidator*.
- __iv. **Save** the process configuration change.
- __h. Modify the validator for the attribute TotalCost.
 - __i. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
 - __ii. Select the Technology Review work item type.
 - __iii. Edit the Total Cost custom attribute.

In the Validators section select the **RegExpScriptedValidator** from the drop down box and press **OK**.



- __iv. **Save** the process configuration change.
- __i. Now the validator needs to be configured with the pattern, modifier, severity and error message.
 - __i. Switch the project area editor to the *Process Configuration Source* tab.
 - __ii. Use CTRL+F and enter **com.acme.providers.script.RegExpScriptedValidation** as your search term.

- __a. The entry you find should contain the data. Please note the XML below has been wrapped to fit the document.

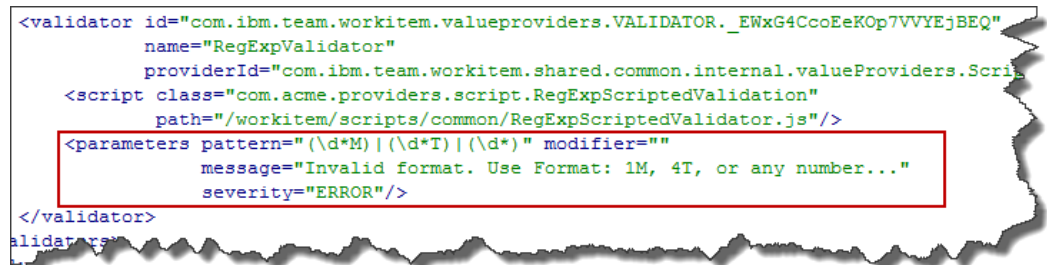
```
<validator id="com.ibm.team.workitem.valueproviders.VALIDATOR._EWxG4CooEeKOp7VVYEjBEQ"
  name="RegExpValidator"
  providerId="com.ibm.team.workitem.shared.common.internal.valueProviders.ScriptAttributeValueProvider">
  <script class="com.acme.providers.script.RegExpScriptedValidation"
    path="/workitem/scripts/common/RegExpScriptedValidator.js"/>
</validator>
```

- __b. Enter the following line with parameters right before the end of the validator section marked with **</validator>**.

```
<parameters pattern="(\d*M)|(\d*T)|(\d*)" modifier=""
message="Invalid format. Use Format: 1M, 4T, or any
number..." severity="ERROR"/>
```

- __c. Review the data you just entered. It is exactly the same information provided in the regular expression validator in the last lab. It contains the [regular expression](#), an empty modifier, the standard message and severity Error.

Your code should look like below.



```
<validator id="com.ibm.team.workitem.valueproviders.VALIDATOR._EWxG4CooEeKOp7VVYEjBEQ"
  name="RegExpValidator"
  providerId="com.ibm.team.workitem.shared.common.internal.valueProviders.Script
  <script class="com.acme.providers.script.RegExpScriptedValidation"
    path="/workitem/scripts/common/RegExpScriptedValidator.js"/>
  <parameters pattern="(\d*M)|(\d*T)|(\d*)" modifier=""
    message="Invalid format. Use Format: 1M, 4T, or any number..."
    severity="ERROR"/>
</validator>
```

- __d. **Save** the process configuration change.
- __j. Create a new work item of type Technology Review. Switch to the Tech Review Details tab. Enter data into the Total Cost attribute.
- __i. Enter **"Foo"**. An error marker should show up and provides the error message you configured as hover.
- __ii. Enter a number or a number followed by 'M' or 'T' and verify the error indicator disappears.

You have just successfully implemented a script based validator that can be configured using XML in the process configuration source. The same configuration technique can be used for all other JavaScript based customization.

5.5 Script based Value Set

Section Scenario



Your project can't provide a web page with the department information to be used with the HTTP Filtered Value Provider at this time. Although everyone agrees it would be the best solution you are asked to replace it with a simple selection of hard coded department names.

You head back to you cubicle, wondering if script based value sets could be a solution and how they would work.

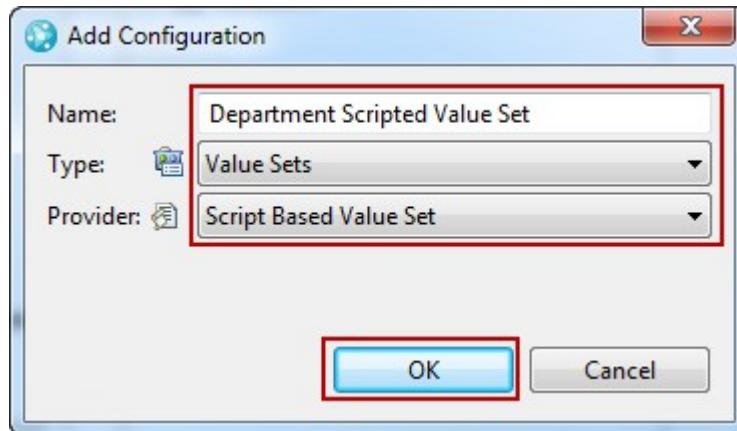
Scripting HTTP Value providers



In [Appendix C](#) we will provide you with information on how to build a script to call the cars service we used as an example in Lab 4.

- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user **jim** password **jim**.
- __2. Open the project area editor for the *Nifty Application Project* project are and switch to the Process Configuration tab.
Open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __a. In the Attribute Customization editor, select *Value Sets*. Use the **Add...** button at the bottom of the editor or use a right click to add a new Value Set.
 - __b. As *Name* enter *Department Scripted Value Set*.
 - __c. As *Type* select *Value Sets*.
 - __d. As *Provider* select *Script Based Value Set*.

- __e. Your Configuration should look as below. Click **OK** to create the configuration.



- __3. To download the example code, click at the **Download example** link in the Configuration editor.
- __i. Browse to the folder
C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\
 - __ii. Change the file name to **ScriptBasedDepartmentValueSet.js**
 - __iii. Press the **Save** button.
 - __iv. **Save** the process configuration.
- __4. To see the new script in Eclipse open the *Project Explorer* view.
- __a. Select the *ScriptBasedCustomization* project
 - __b. Use the *Refresh* context menu entry or press *F5*.
 - __c. The externally saved script file now shows up in Eclipse
 - __d. Open the file for editing.

__e. The example looks like below:

```

* Licensed Materials - Property of IBM
dojo.provide("com.example.ValueSetProvider");

(function() {
    dojo.declare("com.example.ValueSetProvider", null, {

        getValueSet: function(attributeId, workItem, configuration) {

            var result= [];
            result.push("a");
            result.push("b");
            return result;

        }

    });
})();

```

__f. The script pushes values into an array and returns the array

__5. Rename the class and prepare the script for better debugging like before. Debugging is not really needed, the example is not complex, but that might change in the future.

__a. Locate the string "*com.example.ValueSetProvider*" and replace it with "*com.acme.providers.script.ScriptBasedDepartmentValueSet*" to rename the value set.

__b. In the line after the (function() declaration add the lines:

```

var doDebug= true;
var scriptname = "ScriptBasedDepartmentValueSet";

```

__c. Open one of the scripts you created before and copy the debug function over. Insert the code immediately after the statement "return result;".

```

function debug(display) {
    if(doDebug) {
        console.log(scriptname + " " + display);
    }
}

```

__d. Add a debug("- Start"); statement at the beginning of the function as before.

__e. Add a debug("- return results"); statement before the return statement.

__f. Now the script is prepared to allow developing the required capabilities. This time you want to keep the script as configurable as possible.

__6. Add some department values to the script

__a. Change the strings in the existing *result.push()* statements to "*Development*" and "*Sales*".

- __b. Add new `result.push()` statements with the strings "Accounting", "IT", "Production".
- __c. The resulting code should look like below

```

getValueSet: function(attributeId, workItem, configuration) {
    debug("- Start");
    var result= [];
    result.push("Development");
    result.push("Sales");
    result.push("Accounting");
    result.push("IT");
    result.push("Production");
    debug("- return results");
    return result;
}
    
```

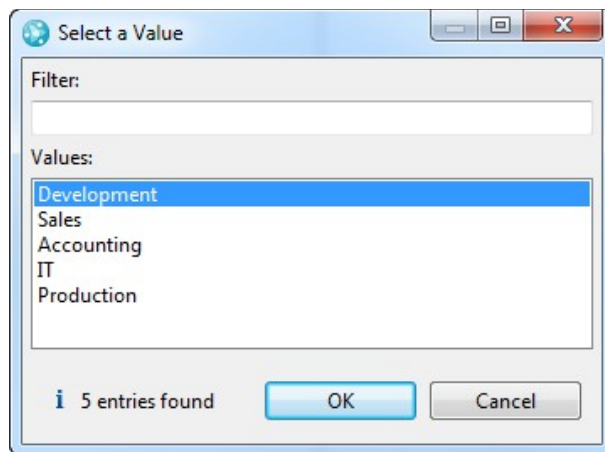
- __d. **Save** the changes to the script.
- __7. Reload the script.
- __a. Go back to the Attribute Customization editor. If you have closed it or navigated away from it, open the *Project Area* editor, switch to the *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section. Locate the customization type *Value Set*, expand its node and click at the customization.
 - __b. In the Configuration editor section press **Reload** to upload the changes.
 - __c. Your configuration should look as follows. Please note, keep the Filtered check box in the Value Set Provider Script Settings unchecked.

The screenshot shows a configuration interface with the following fields:

- Details:**
 - Name: Department Scripted Value Set
 - Provider: Script Based Value Set
- Configuration:**
 - Local File Path: C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\ScriptBasedDej [Reload] [Browse...]
 - Attachment Path: /workitem/scripts/common/ScriptBasedDepartmentValueSet.js [Save As...]
 - Class Name: com.acme.providers.script.ScriptBasedDepartmentValueSet
 - Example Code: [Download example](#)
- Value Set Provider Script Settings:**
 - Filtered

- __d. **Save** the Process configuration change.

- __8. Configure the Attribute for the new Attribute Customization
- __a. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
 - __b. Select the *Technology Review* work item type. Scroll down to the *Attributes* section.
 - __c. In the *Attributes* section check the **Show only custom attributes** check-box to narrow down your search.
 - __d. Locate and select the custom attribute *Affected Departments*. Press the **Edit** button.
 - __e. Use the drop down button *Value Set* to select the **Department Scripted Value Set**.
 - __f. Click **OK** and save the changes to the process configuration.
- __9. Test your new Value Set.
- __a. Create a new work item of Type Technology Review.
 - __b. Open the Tech Review Details Tab.
 - __c. Press the **Add...** button on the Affected Departments attribute.
 - __d. You see a value selection dialog like below.




- __e. Select the values *Development* and *IT* and click **OK**.
- __f. The selected values are added to the Affected Departments list.
- __g. Press the **Add...** button again.
- __h. In the selection dialog select the *Filter* field and *type* something for example 'A'.

- __i. The filter does not work. To have the filter working follow the info box below.

JavaScript Based Value Sets and Filtering

Please read at [Attribute customization Wiki entry](#) as well as the [special Wiki entry around Value Set Providers](#).

To allow your script based value provider to use filtering, access the filter it needs to implement the interface



```
getFilteredValueSet :
function(attributeId, workItem,
configuration, filter)
```

This interface provides access to the filter and can act on it.

- __ii. The example script ***ScriptBasedDepartmentValueSetFiltered.js*** which is part of the script sources that you can download from the lab material provides you with an example. It creates a regular expression pattern that tests for a matching prefix and can be used as type ahead filter.

- __10. If you have issues with the script, you can debug it.

- __a. To debug your script open the script log of the server at
C:\JSWorkshops\IBM\JazzTeamServer\server\tomcat\work\Catalina\localhost\ccm\eclipse\workspace\metadata\log

Script Log file for Value Sets

Value Sets as used by the 'Value Set Combo' and the 'Value Set Picker' presentation (typically HTTP value set providers) are only evaluated on the server and therefore always log into the server log



```
<JazzServerInstallDir>\server\tomcat\work\Catalina\localhost\ccm\eclipse\workspace\metadata\log
```

In case of a 2.x context root, replace ccm by jazz in the string above.

In this workshop <JazzServerInstallDir> resolves to
C:\JSWorkshops\IBM\JazzTeamServer

You have just successfully implemented a script based value set.

5.6 Calculated Value to Visualize the State of the Technology Review

Section Scenario



Your project has discovered that management does not understand the state of the Technology Review Work Items. During the last meeting you have been asked if it would be possible to display the current state and the valid actions as a graphic.

You head back to your cubicle, wondering if you will be able to fulfill this request and how that could work.

This Solution Uses an Internal Service



The solution described below relies on the attachment service which is internal and not documented. There are currently no plans to change the service. If the service changes this feature might stop working.

You can skip this section if you want to, but it provides some interesting additional knowledge.

- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace `C:\JSWorkshops\Workspaces\Lab1`. Log in with the user `jim` password `jim`.
- __2. Review the Technology Review work item.
 - __a. Create a work item of the type *Technology Review*.
 - __b. Switch over to the *Tech Review Details* tab.
 - __i. The tab contains a blank section named *Workflow Information* that you probably already noticed.

Tech Review Miscellaneous

Technology Name:

Workflow Information:

- __ii. The Workflow Information section is a read only Wiki type attribute that can be used to display information such as links.

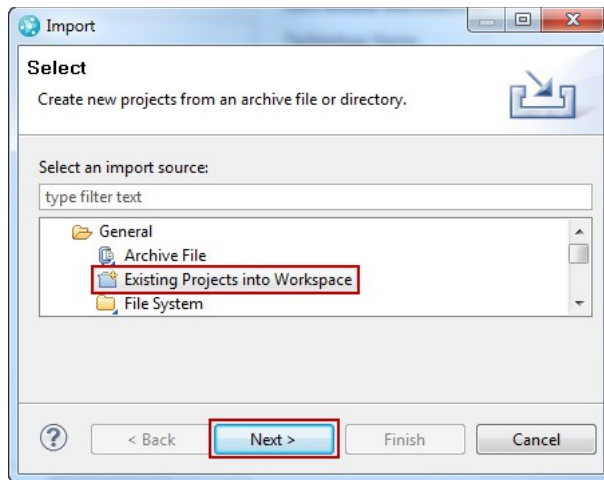
- ___iii. The Wiki also displays certain resources provided by a REST service automatically. You will use this mechanism to display the workflow of the work item.

___3. Download the supporting files

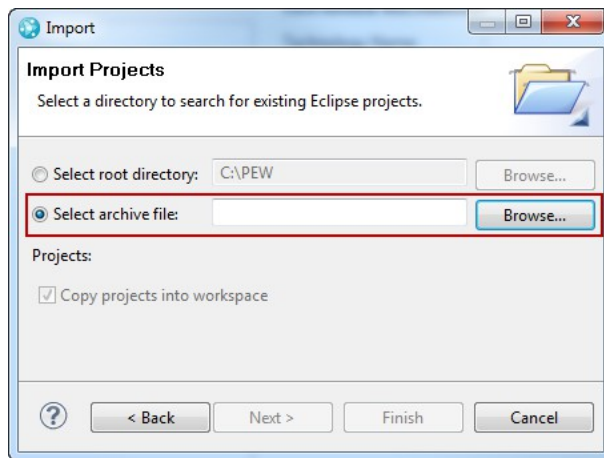
- ___a. Browse to the [download location of this workshop](#). In the Lab Material section, locate the link **Lab 5.6 Workflow Image Default Value**. Download the referenced file and store the file on your local drive and remember the location. The file on disk will be named **YYYYMMDD_ScriptBasedCustomization_Lab5_WorkFlowImage_Files.zip**. The prefix is of the form YYYYMMDD, where YYYY, MM, DD represent year, month of year and day of month.

___4. Use the Eclipse menu **File > Import** to open the import wizard.

- ___a. In the import wizard select **Archive File** in the category **General**.

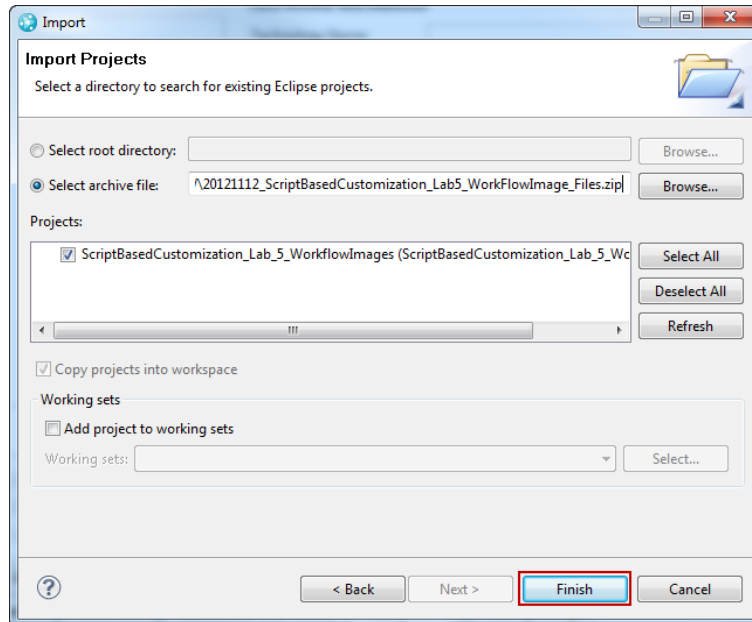


- ___b. Use the **Next** button.
- ___c. Select the radio-button **Select Archive File**.

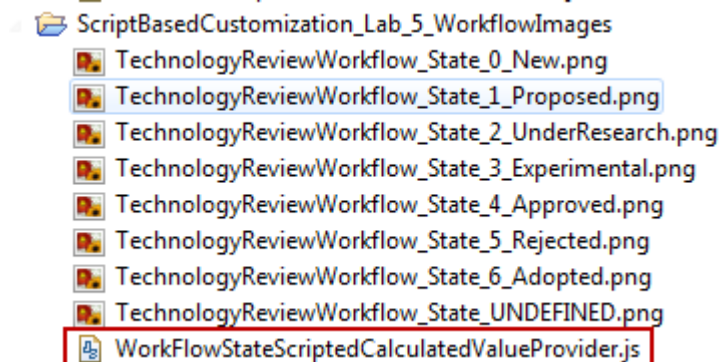


- ___d. Browse to the file you downloaded, select it and press **Open**.

- __e. The screen should show a new project for import.



- __f. Press **Finish** to import the files.
- __g. Open the Eclipse Project Explorer view. You should now see a new project in your Eclipse workspace. The project contains several image files and a JavaScript file.



- __5. Open and review the JavaScript file.
- __a. Open the JavaScript file in an editor, for example in the JavaScript or text editor in Eclipse.

- __b. The file looks similar to scrips you have already developed. It uses the debug output capabilities, it looks at the state value of a work item. The main code is a decision table that returns some text based on the value of the workflow state

```

getValue: function(attribute, workItem, configuration) {
    debug("Start");

    var workflowState = workItem.getValue(WorkItemAttributes.STATE);
    debug("WorkflowState: " + workflowState);

    // Replace <URL> with the resource URL to the image attached to a work item
    if(workflowState=="") return "{<URL> New}";
    if(workflowState=="tecReviewWorkflow.state.s1") return "{<URL> Proposed}";
    if(workflowState=="tecReviewWorkflow.state.s2") return "{<URL> Under Research}";
    if(workflowState=="tecReviewWorkflow.state.s3") return "{<URL> Experimental}";
    if(workflowState=="tecReviewWorkflow.state.s4") return "{<URL> Approved}";
    if(workflowState=="tecReviewWorkflow.state.s5") return "{<URL> Rejected}";
    if(workflowState=="tecReviewWorkflow.state.s6") return "{<URL> Adopted}";
    return "{<URL> Unknown State}";

    function debug(display) {
        if(doDebug) {
            console.log(scriptname + " " + display);
        }
    }
}

```

- __c. The return statement below will return a URL and a state name as text. The place holder <URL> needs to be replaced by a real URL. The statement below works for the state Proposed.

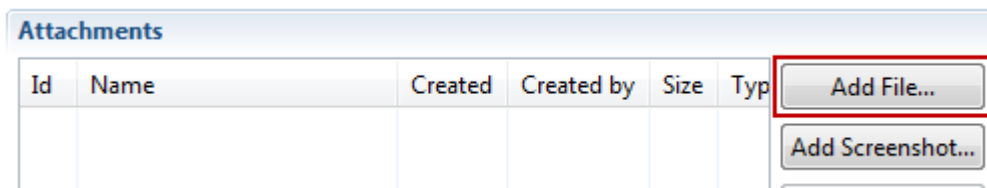
```
return "{<URL>|Proposed}";
```

- __d. The values of the State ID's have been identified by looking them up in the Process Configuration source as described in 5.3 Script Based Conditions section __7. __a. - __d. on page 157-158.

- __6. Prepare the workflow images so that they can be displayed in the Wiki.

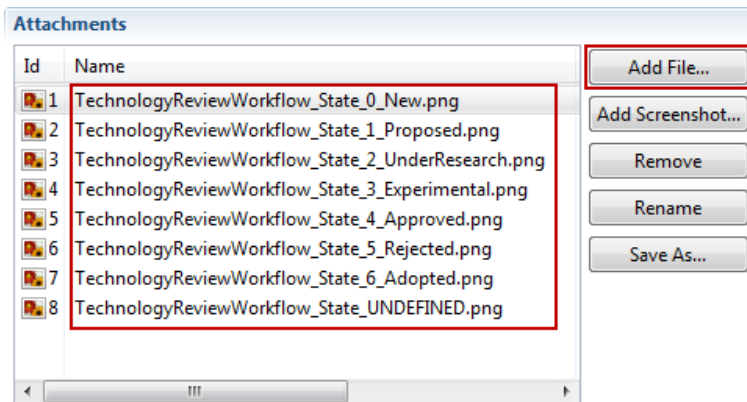
Rational Team Concert does not display all data provided by a link in the Wiki. It does however display images that are defined as a work item attachment.

- __a. Create a work item in which to store the attachments. Create a **Task** and name it “*DO NOT DELETE ME - SERVING REFERENCED ATTACHMENTS*”
- __b. Switch to the *Links* tab.
- __c. In the Attachments section use the button **Add File...**

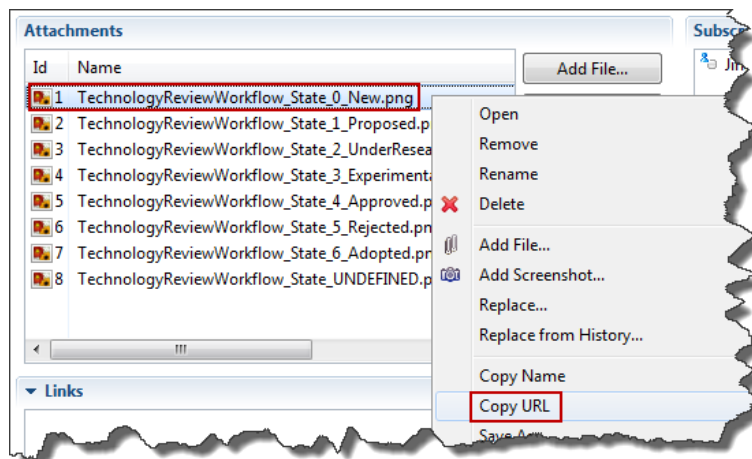


- __d. Browse to the *ScriptBasedCustomization_Lab_5_Workflow/Images* folder in *C:\JSWorkshops\Workspaces\Lab1*.

- __i. Select the first image ***TechnologyReviewWorkflow_State_0_New.png***.
 - __ii. Click **Open** to upload the image file.
- __e. Upload all other images in the order of the file names.
- __i. In the *Attachments* section use the button **Add File....**
 - __ii. Browse and select the next file
 - __iii. Click **Open** to upload the image file.
- __f. Repeat the steps i-iii for all remaining files keeping the order of the files.
- __g. **Save** the work item.
- __7. get the URL for an attachment and add it to the script.
- __a. The work item should now contain all the images for all the state, including an image for an undefined state.



- __b. Right click on the attachment ***TechnologyReviewWorkflow_State_0_New.png*** and select copy URL.



- __c. Switch to the editor for the JavaScript file
WorkflowStateScriptedCalculatedValueProvider.js.

- __i. Search the line for the state **New**, mark the **<URL>** string and replace it with the value of the URL you copied in the previous step.

```
// Replace <URL> with the resource URL to the image attachment
if(workFlowState=="") return "{{<URL>|New}}";
if(workFlowState=="tecReviewWorkflow.state.s1") return "
if(workFlowState=="tecReviewWorkflow.state.s2") return "
```

- __ii. The line should now look similar to

```
if(workFlowState=="") return
"{{https://clm.process.ws/ccm/service/com.ibm.team.workitem.
common.internal.rest.IAttachmentRestService/itemName/com.ibm.
team.workitem.Attachment/1|New}}";
```

- __iii. The basic URL is a common URL that is used for all attachments. The individual attachments are accessed by their number.

- __d. Replace the rest of the **<URL>** strings with valid attachment URL's.

- __i. Replace all the **<URL>** strings left by the following code

```
https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.t
eam.workitem.Attachment/
```

and leave the number out.

- __ii. Look up the Attachment numbers in the work item for all images. It is likely a consecutive list.

- __iii. Insert the attachment number at the end of the URL's.

- __e. Your code should now look similar like below.

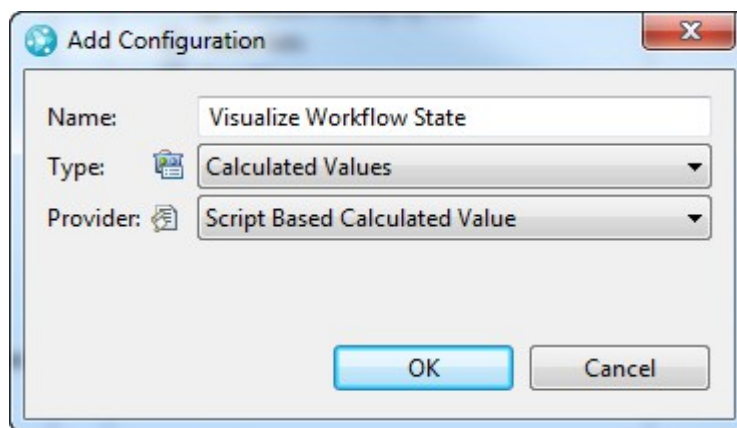
```
// -> with the resource URL to the image attachment
if(workFlowState=="") return "{{https://clm.process.ws/ccm//service/com.ibm.team.workitem.
common.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/1|New}}";
if(workFlowState=="tecReviewWorkflow.state.s1") return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/2|Prop
if(workFlowState=="tecReviewWorkflow.state.s2") return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/3|Unde
if(workFlowState=="tecReviewWorkflow.state.s3") return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/4|Expe
if(workFlowState=="tecReviewWorkflow.state.s4") return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/5|Appi
if(workFlowState=="tecReviewWorkflow.state.s5") return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/6|Rej
if(workFlowState=="tecReviewWorkflow.state.s6") return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.co
mmon.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/7|Ador
return "https://clm.process.ws/ccm//service/com.ibm.team.workitem.common.internal.rest.IAttachmentRestService/itemName/com.ibm.team.workitem.Attachment/8|Unknown State}}";
```

- __f. The last line in the script will show an image if no matching state can be found. This adds an indicator to the users, that the process has been changed and the images are not correct anymore.

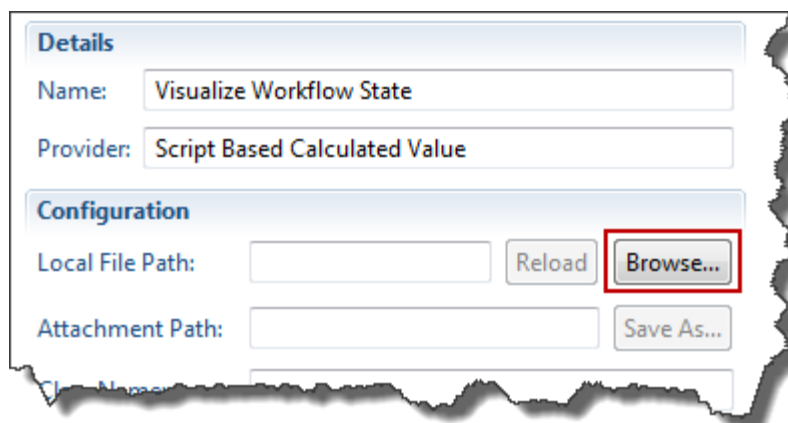
- __g. **Save** your changes to the JavaScript code.

- __8. Now you can configure your JavaScript based default value provider to test it. Switch to the Process Configuration tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
- __a. Use the **Add...** button to add a Calculated Value
 - __b. As *Name* enter **Visualize Workflow State**
 - __c. Select the *Type* **Calculated Values**
 - __d. Select **Script Based Calculated Value** as *Provider*.

The configuration dialog should look like below:



- __e. Press **OK** to set the configuration.
- __f. In the *Configuration* editor section to the right use the **Browse...** button to browse for your script.



- __i. Browse to your folder:

C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization_Lab_5_Workflow\images

__ii. Select the Script below and use the *OK* button.

WorkflowStateScriptedCalculatedValueProvider.js

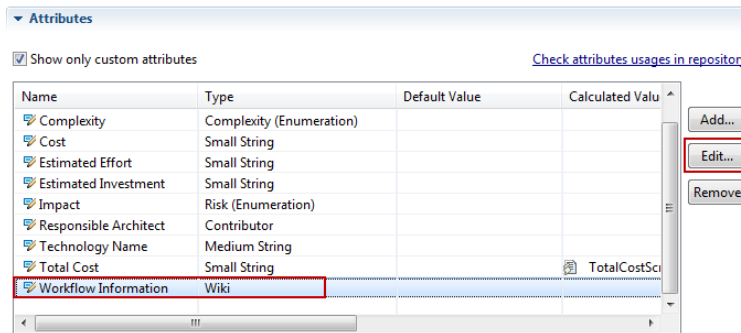
__iii. **Save** the changed process configuration.

__9. Configure the attribute to use the JavaScript Based Calculated Value. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.

__a. Select the *Technology Review* work item type. Scroll down to the *Attributes* section.

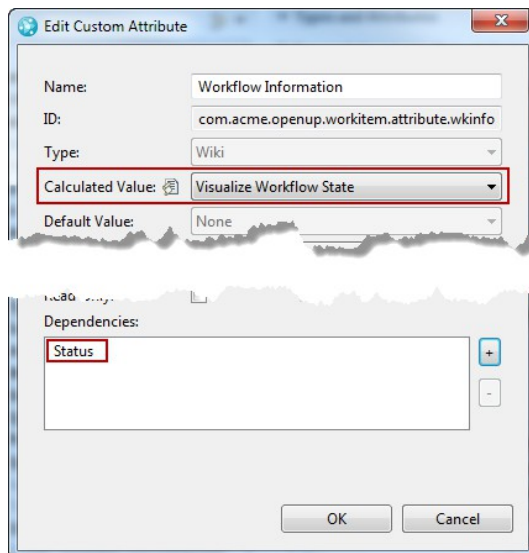
__b. In the Attributes section check the **Show only custom attributes** check-box to narrow down your search.

__c. Locate and select the custom attribute *Workflow Information*. Press the **Edit** button.



__d. In the Attribute Editor use the *Calculated Value* drop down button and select **Visualize Workflow State**.

__e. Add a dependency to the attribute **Status**. Select **OK**.



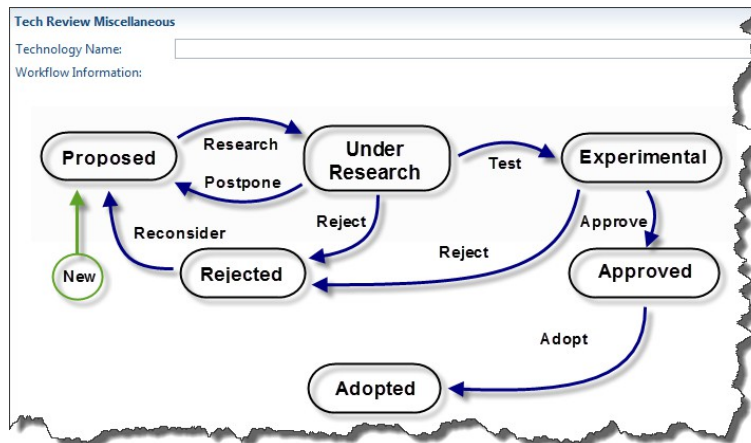
__f. **Save** the change to the process configuration.

__10. Test your work.

__a. Create a work item of type *Technology Review* in the Eclipse Client.

__b. Switch to the *Tech Review Details* tab

__i. The workflow Information should be visible as below.



__ii. Provide a *Summary* and other attributes as needed and **save** the work item.

__iii. Now it shows the Proposed state.

__iv. Change the states to check that the images match the states.

__c. Create a work item of type *Technology Review* in the Web Client.

__i. Switch to the *Tech Review Details* tab.

__ii. The Workflow Information shows text for the new state. Use the **Preview** button to see the current value.

Tech Review Miscellaneous

Technology Name:

Workflow Information:

Preview

__iii. Save the work item. Add required attributes if necessary. The Workflow information now displays the *Proposed* state.

__iv. Move the work item through the states to confirm all is working as desired.

You have successfully created a calculated value provider that displays the state machine as graphics.

5.7 Summary

You have successfully created JavaScript based attribute customization. This customization allows more flexibility as your process matures and provides you with extended capabilities to compute:

- Default Values
- Attribute Conditions
- Attribute Validators
- Value Sets

5.8 Solutions

You can download example solutions for this lab from the workshop download page. The solutions are available as Link **Lab 5 Solution Files**.

- __1. Browse to the [download location of this workshop](#).
 - __a. In the section Lab Material locate the link **Lab 5 Solution Scripts**.
 - __b. Download the referenced file and store the file on your local drive and remember the location. The file on disk will be named **YYYYMMDD_ScriptBasedCustomization_Lab5_Solutions_Scripts.zip**. The prefix is of the form YYYYMMDD, where YYYY, MM, DD represent year, month of year and day of month.
- __2. Use the Eclipse menu **File>Import** to open the import wizard. Import the solution files following the steps in 5.6__4. on page 182.

5.9 Appendix A - Script Troubleshooting

Writing JavaScript can be challenging. The lack of strict types and a challenging debugging environment are the main contributors to this situation.

Here some hints derived from issues that came up while creating the material for this Lab

- Use a lot of logging.
- Use a text editor that does not lock the log file, detects changes to the log file and reloads the log file if requested. One good specimen of this type of text editor is [Notepad++](#).
- If the client log does not contain information about your script, check the server log.
- Don't forget to reload and save your process configuration after changes to the scripts
- Close work item editors and re-open them to make sure the new script version is used in the work item.
- If you are not confident of the uploaded version of the script delete the script attachment from the process attachments.
- Variables and functions that get called cause `com.ibm.team.repository.common.TeamRepositoryException: Unexpected exception type, if they don't exist.`
- Missing Brackets or semicolons throw weird errors such as “missing ; before statement”.
- You can surround your code with throw/catch blocks in order to make it more resilient.

5.10 Appendix B – Script Debugging

5.10.1 Script debugging example with Chrome

Please see this [article about debugging scripts with Chrome](#).

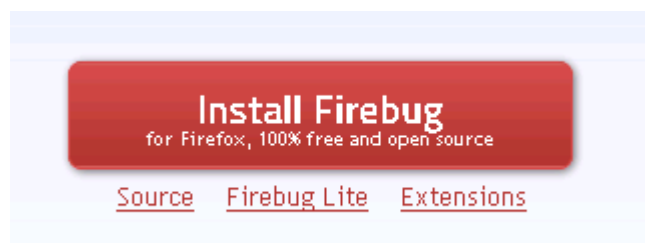
5.10.2 Script debugging example with Firebug

This appendix section shows you an example of debugging one of the scripts in this lab using Firebug as an alternative option to the code logging statements that you have been coding in the lab examples. Note that we give some basic hints on how to use Firebug Firefox add-on, but you may want to use other debuggers of your choice that provides you with similar features.

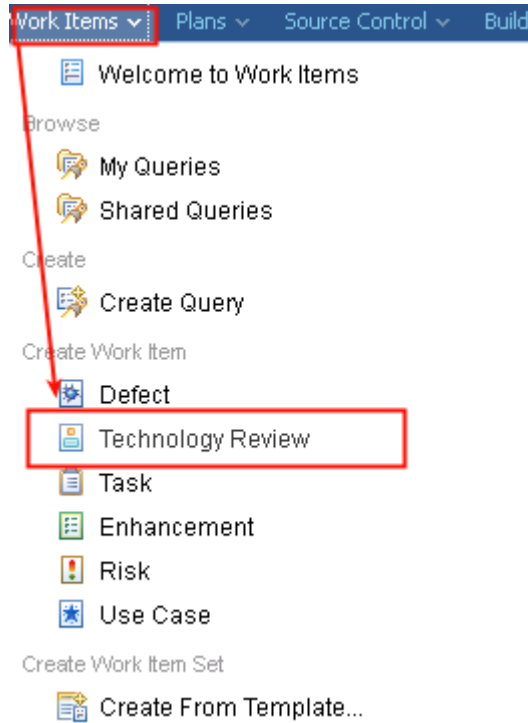
The script we use in this example is the one from the [Section 5.2.1.Total Cost Calculated Value](#). It is assumed that you have already completed that section so the script is already working as per lab instructions.

The steps to follow are:

- __1. Open Firefox browser. If you don't have Firefox already installed on your system you can get it from this [site](#). Try to install a version that follows the Rational Team Concert system requirements.
- __2. Within Firefox, navigate to the [Firebug](#) page, click the **Install Firebug** button and follow the instructions. Your browser will be restarted after installation.



- __3. In Firefox, navigate to Rational Team Concert, to the Nifty Application Project project area. For example, following the lab convention, you can directly navigate to: <https://clm.process.ws/ccm/web/projects/Nifty%20Applicaition%20Project>
- __4. Log in with the user **jim** password **jim**.
- __5. Create a new work item of type *Technology Review*
Note you can also open an existing work item.



- ___a. If you create a new work item, give it a summary and save it.
- ___b. Modify the url in your browser to add more information for debugging the script including the “?debug=true”. For example:

<https://clm.process.ws/ccm/web/projects/Nifty%20Applicaition%20Project?debug=true#action=com.ibm.team.workitem.viewWorkItem&id=<YourCurrentWIID>>

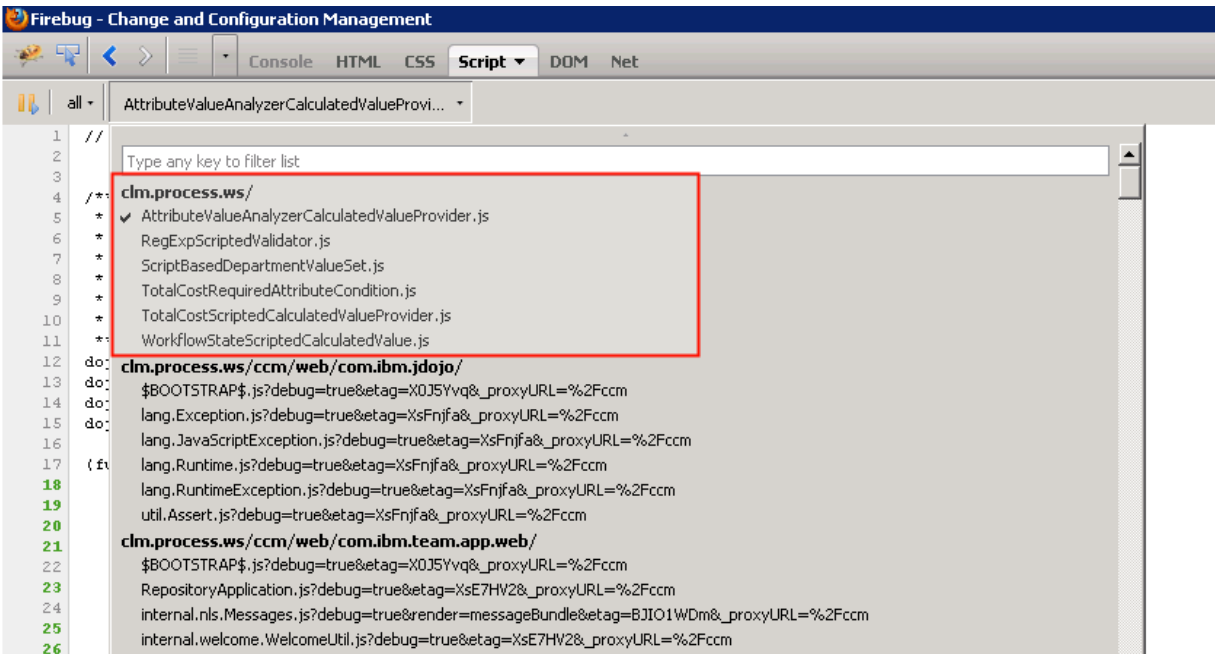
Note that adding the “debug=true” parameter is not mandatory, but it will remove some code compression and provide more information available for debugging

- ___6. Open Firebug clicking **Tools > Firebug > Open Firebug**, or clicking the icon in your browser



- ___7. Switch to the Script tab, and click the **Enable** link to activate that view. You may be asked to reload the page for it to begin working.
- ___8. Check the script has been loaded:

- ___a. Open the drop-down list that appears at the top of the Firebug Script view
- ___b. You should find the script listed under the root of your server Public URI, in this case as sub-nodes under "clm.process.ws"



- ___c. Click TotalCostScriptedCalculatedValueProvider.js script to make the code appear in the debugging section.

___9. Add a breakpoint:

- ___a. Click in the left margin of the code line where the value of the Estimated Invest is gathered:

```
var estimatedInvest =
workItem.getValue("com.acme.openup.workitem.attribute.estinvest");
```

- ___b. You should see a red circle that denotes the breakpoint something similar as follows:

```
11
12 dojo.provide("com.acme.providers.script.TotalCostScriptedCalculatedValue");
13 dojo.require("com.ibm.team.workitem.api.common.WorkItemAttributes");
14
15 (function() {
16     var doDebug = true;
17     var scriptname = "TotalCostScriptedCalculatedValue";
18     dojo.declare("com.acme.providers.script.TotalCostScriptedCalculatedValue", null, {
19
20         getValue: function(attribute, workItem, configuration) {
21
22             debug("- Start");
23             var estimatedInvest = workItem.getValue("com.acme.openup.workitem.attribute.estinvest");
24             console.log("Estimated Invest: " + estimatedInvest);
25
26         }
27     });
28 }
```

__10. Debug the code:

__a. Back in the Technology Review work item introduce a value in one of the attributes that will make the script fire, for example Estimated Effort

__b. Firebug will get focus high-lighting the line with the breakpoint

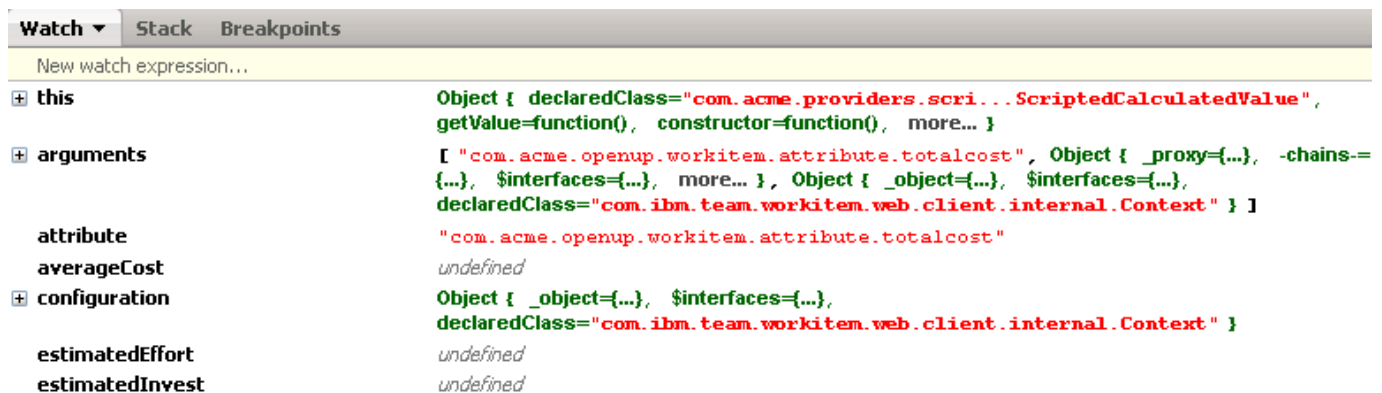
```

20     getValue: function(attribute, workItem, configuration) {
21
22         debug("- Start");
23         var estimatedInvest = workItem.getValue("com.acme.openup.workitem.attribute.estinvest");
24         console.log("Estimated Invest: " + estimatedInvest);
25
26         var estimatedEffort =workItem.getValue("com.acme.openup.workitem.attribute.esteffort");
27         debug("Estimated Effort: " + estimatedEffort);
28     }

```

__c. The right side of the window provides lots of valuable information with the usual views you find in any debugger.

__i. The *Watch* section may be the most useful sections when you are beginning to develop scripts like the ones in this lab. This is because uou are probably not familiar yet with the information you have access to or how to access it in work items.
 You can also add expressions in this view to further debug your code or test new code for new features you want to add to your scripts.



__d. On top of the left side of the debug window you have access to the typical controls for controlling the debug code flow:



These are the basics steps to begin using the Firebug browser add-on for debugging your scripts code. Once you are done with your code debugging, remember to remove the “debug=true” parameter from your url, and to shutdown Firebug to reduce resource consumption.

5.11 Appendix C – Scripted HTTP Value Set Provider

This appendix will show you how you can use a scripting attribute customization that will call an external service to populate the results. The example uses the same service used in Lab 4 for the HTTP Filtered Value Set (section 4.2.2). The example is placed in this appendix section because of the lack of a service with representative values that could be used with the lab scenario.

More information and code samples

The code in this appendix is based on the information you can find in the jazz.net wiki page called. [Tutorial for providing a custom value set provider](#).



You can find the code described below in the workshop download files in a JavaScript source file called "DepartmentProviderScripted.js".

You can also find information about using this feature to gather information from the OSLC services of your Jazz deployment in this [blog entry](#)

The steps to follow are:

- __1. Open the Eclipse client if it is not already opened and connect to your repository. You can use the workspace C:\JSWorkshops\Workspaces\Lab1 . Log in with the user **jim** password **jim**.
- __2. Open the project area editor for the *Nifty Application Project* project area and switch to the Process Configuration tab.
Open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section.
 - __a. In the Attribute Customization editor, select *Value Sets*. Use the *Add* button at the bottom of the editor or use a right click to add a new Value Set.
 - __b. As *Name* enter *Department Provider Scripted*
 - __c. As *Type* select *Script Based Value Set*.
 - __d. As *Provider* select *Script Based Value Set*.
- __3. To download the example code, click at the **Download example** link in the Configuration editor.
 - __i. Browse to the folder
C:\JSWorkshops\Workspaces\Lab1\ScriptBasedCustomization\
 - __ii. Change the file name to **DepartmentProviderScripted.js**

- __iii. Press the **Save** button.
- __iv. **Save** the process configuration.

__4. To see the new script in Eclipse open the *Project Explorer* view.

- __a. Select the *ScriptBasedCustomization* project
- __b. Use the *Refresh* context menu entry or press *F5*.
- __c. The externally saved script file now shows up in Eclipse
- __d. Open the file for editing.
- __e. The example looks like below:

```

* Licensed Materials - Property of IBM
dojo.provide("com.example.ValueSetProvider");

(function() {
    dojo.declare("com.example.ValueSetProvider", null, {

        getValueSet: function(attributeId, workItem, configuration) {

            var result= [];
            result.push("a");
            result.push("b");
            return result;

        }

    });
})();

```

- __f. The script pushes values into an array and returns the array

__5. Rename the class and prepare the script for better debugging like before.

- __a. Locate the string *"com.example.ValueSetProvider"* and replace it with *"com.acme.providers.script.DepartmentProviderScripted"* to rename the value set.
- __b. In the line after the *function()* declaration add the lines:

```

var doDebug= true;
var scriptname = "DepartmentProviderScripted";

```

- __c. Open one of the scripts you created before and copy the debug function over. Insert the code below right behind the statement *return result*.

```

function debug(display) {
    if(doDebug) {
        console.log(scriptname + " " + display);
    }
}

```


__6. Prepare the script to use the HTTP connector:**__a.** Add the following dojo.require statement after the dojo.provide one:

```
dojo.require("com.ibm.team.workitem.api.common.connectors.HttpConnectorParameters");
```

That dojo class provides the needed features to open a connection and manage the retrieval of values to call an external service.

__b. Add a shortcut to the HTTPConnectorParameters to ease the development. To do so, add the following statement after the function() declaration:

```
var HttpConnectorParameters =
com.ibm.team.workitem.api.common.connectors.HttpConnectorParameters;
```

__c. The customized code looks like the following so far:

```
dojo.provide("com.acme.providers.script.DepartmentProviderScripted");
dojo.require("com.ibm.team.workitem.api.common.connectors.HttpConnectorParameters");

(function() {

    var HttpConnectorParameters = com.ibm.team.workitem.api.common.connectors.HttpConnectorParameters;

    var doDebug= true;
    var scriptname = "DepartmentProviderScripted";

    dojo.declare("com.acme.providers.script.DepartmentProviderScripted", null, {
```

__7. Add the parameters and the call the service:**__a.** First you need to build an object that will hold the parameters for the service call. Add the following code after the getValueSet function declaration statement:

```
var params= new HttpConnectorParameters();
    params.url="http://cars.flashmx.us/makes";
    params.xpath= "//xml/node";
    params.columnXpaths= ["/make"];
    params.columnIds= ["Department"];
```

__b. To call the service with the provided parameters, add the following code:

```
var connector=  
configuration.getDataConnector("HttpConnector");  
var values= connector.get(params);
```

- ___c. If the parameters are well set and the call has succeeded, you will have the response for the service in the values variable. The result array will be returned with the desired data.

Note that from this code template you can do whatever post-processing on the data you wish to do to customize the values to be presented to the user.
In this example you will perform the same formatting as you did in Lab 4.

Add the following code before the return statement, replacing the 2 lines after *var result=[]*; with this code:

```
while(values.hasNext()){  
    var entry= values.next();  
    var Dept= entry.getId("Department") + " Department";  
    result.push(Dept);  
}
```

- ___d. The script code for the `getValueSet()` function should look like the following. **Save** your changes.

```

dojo.declare("com.acme.providers.script.DepartmentProviderScripted", null, {

    getValueSet: function(attributeId, workItem, configuration) {

        var result= [];

        var params= new HttpConnectorParameters();
        params.url="http://cars.flashmx.us/makes";
        params.xpath= "//xml/node";
        params.columnXpaths= ["/./make"];
        params.columnIds= ["Department"];

        var connector= configuration.getDataConnector("HttpConnector");
        var values= connector.get(params);

        while(values.hasNext()){
            var entry= values.next();
            var Dept= entry.getId("Department") + " Department";
            result.push(Dept);
        }

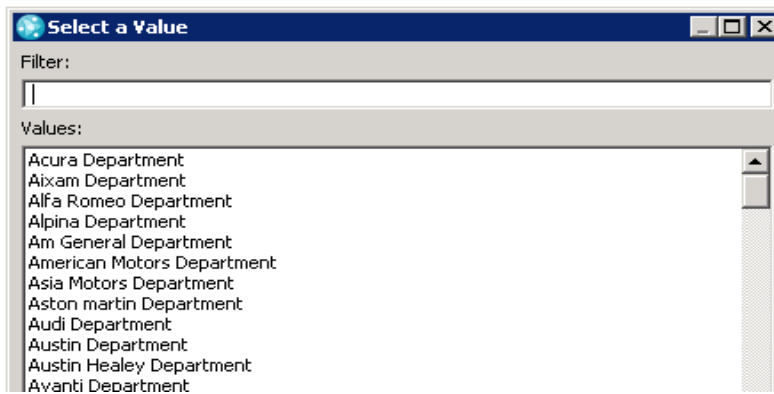
        return result;

        function debug(display){
            if(doDebug){
                console.log(scriptname + " " + display);
            }
        }
    }
}

```

- __8. Reload the script.
 - __a. Go back to the Attribute Customization editor. If you have closed it or navigated away from it, open the *Project Area* editor, switch to the *Process Configuration* tab and open the **Configuration > Project Configuration > Configuration Data > Work Items > Attribute Customization** section. Locate the customization type *Value Set*, expand its node and click at the customization.
 - __b. In the Configuration editor section press **Reload** to upload the changes.
 - __c. **Save** the Process configuration change.
- __9. Configure the Attribute for the new Attribute Customization
 - __a. Open the **Configuration > Project Configuration > Configuration Data > Work Items > Types and Attributes** section.
 - __b. Select the *Technology Review* work item type. Scroll down to the *Attributes* section.
 - __c. In the *Attributes* section check the “*Show only custom attributes*” check-box to narrow down your search.

- __d. Locate and select the custom attribute *Affected Departments*. Press the *Edit* button.
 - __e. Use the drop down button *Value Set* to select the **Department Provider Scripted**
 - __f. Click **OK** and save the changes to the process configuration.
- __10. Test your new Value Set.
- __a. Create a new work item of Type Technology Review.
 - __b. Open the Tech Review Details Tab.
 - __c. Press the Add... button on the Affected Departments attribute.
 - __d. You see the values retrieved from the service and presented formatted as below



This appendix section has shown you how the same example that we configured in Lab 4 to call an external service, can be also reproduced using the scripting technologies that you learned in this lab. This specific case has lots of potential uses to integrate information sources in your enterprise where the service call needs some sort of customization based. For example, to address work items attribute values or for returning results needed when post-processing complex computation.

5.12 Appendix D – Changes in 4.0.3 attribute customization script based editor

This lab was originally developed and tested with Rational Team Concert 4.0 and 4.0.0.1 versions. If you are running the lab with a 4.0.3 or greater version, you will find some differences in the configuration editor for Script Based Attribute Customization. The new configuration editor looks like the following:

The screenshot shows a configuration editor with two main sections: 'Details' and 'Configuration'.
In the 'Details' section, there are two input fields: 'Name:' with the value 'TotalCostScriptedCalculatedValue' and 'Provider:' with the value 'Script Based Calculated Value'.
The 'Configuration' section has a warning icon and the text 'No class declaration found'. Below this, there are three input fields: 'Attachment Path:' containing '/workitem/scripts/common/totalcostscriptedcalculatedvalue.js', 'Class Name:', and 'Script:'. To the left of the 'Script:' field is a blue link labeled 'Fill in example'. To the right of the 'Script:' field are three buttons: 'Load...', 'Save As...', and 'Revert'. The bottom of the 'Script:' area is decorated with a sawtooth pattern.

The main differences you have to keep in mind while following the instructions of this lab in a 4.0.3 environment are:

-
- When you are instructed to download the script sample and modify it: you have to first click the **Fill in example** hyperlink. The sample scripts contents will be loaded in the in place editor window and then you can either:
 - Directly follow the lab instructions filling contents in the in place editor
 - Click **Save As** to edit it from your eclipse workspace.

-
- When you are instructed to click **Reload**:
 - No reload is needed if you edited the script using the in place editor
 - Otherwise, you will need click **Load** to upload the new contents from the file system. Then you can **Save** your process changes

You can find additional details in the blog post [Script based attribute customization in RTC 4.0.3: configuration editor improvements](#).

Appendix A Glossary

Term	Definition
Practice (specific to JTS)	
Practice (specific to RMC)	
Process Definition	
Process Description (general)	
Process Description (specific to RTC)	
Process Enactment	
Process Nature	
Process Template	
Project Area	
Published Process	
Rational Method Composer (RMC)	
Rational Team Concert (RTC)	
Task Descriptor	
Work Item Template	

Appendix B 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:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, 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.

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 and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix C Trademarks and copyrights

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

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	Tivoli	WebSphere	Workplace	System p	

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. See Java Guidelines

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

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



© Copyright IBM Corporation 2012

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

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



Please Recycle