Tivoli software

IBM

Redbooks Paper

Edson Manoel
Kartik Kanakasabesan

Deploying Applications Using IBM Rational ClearCase and IBM Tivoli Provisioning Manager

Introduction

This IBM® Redpaper presents a simplified customer environment in which we demonstrate a case study for provisioning applications developed using IBM Rational ClearCase.

This paper focuses on the tasks required for planning, implementing, and configuring all components of IBM Tivoli Provisioning Manager for dynamic provisioning of applications directly from the IBM Rational ClearCase software repository to a target server. The following topics are discussed:

Scenario overview

In our scenario, we will demonstrate one of the methods and requirements for integrating an application staged in IBM Rational ClearCase into IBM Tivoli Provisioning Manager. We show the ability to use IBM Tivoli Provisioning Manager to deploy an application that has been developed and staged in a IBM Rational ClearCase environment. We are going to demonstrate a scenario in which a fixed version of an application is applied to an existing application infrastructure using IBM Tivoli Provisioning Manager. The application will be developed in the IBM Rational ClearCase environment and staged for IBM Tivoli Provisioning Manager to be provision to the existing application environment.

Our fictitious customer is running an Auction Application on a Apache Tomcat, and during the course of day-to-day business operations, the customer notices several problems with the application and has the development team addressing the problems. This process triggers several development activities to address the problem which are as follows:

- Fixing the code problem
- Building the code fix and testing it to make sure it works
- Packaging the built artifacts to stage for IBM Tivoli Provisioning Manager to deploy in to various environments (i.e., Test, Pre-Production and Production environments).

The current production infrastructure for the application supported by the customer is highlighted in Figure 1 on page 3. Currently application provisioning from development is managed manually to maintain the environment shown in Figure 1 on page 3. later in this paper, we show all the required steps to automate the deployment of this sample application using IBM Tivoli Provisioning Manager.

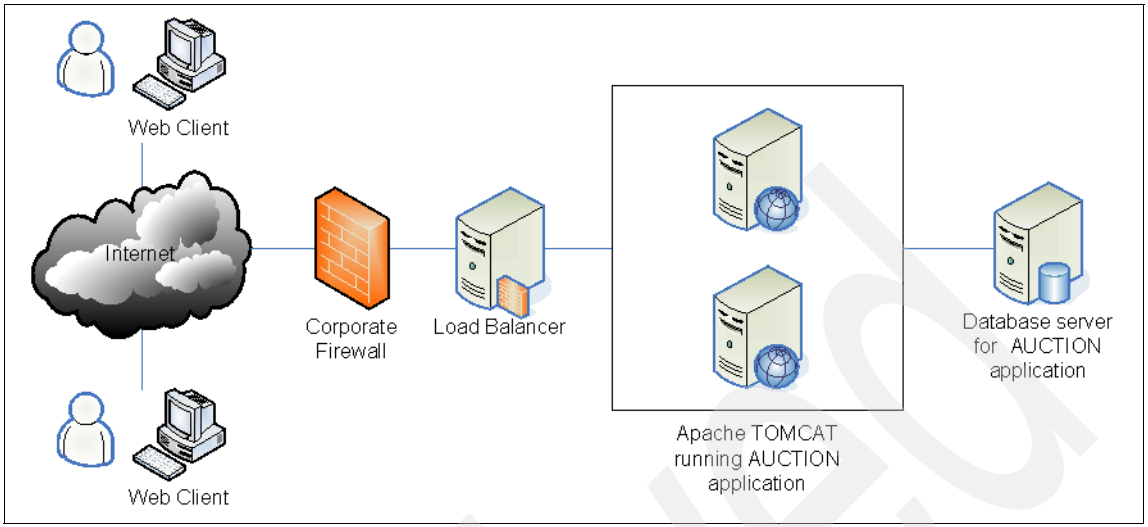

Figure 1 Application infrastructure

In this paper we show the relevant steps needed to integrate the IBM Rational ClearCase development environment of our sample Auction application to the IBM Tivoli Provisioning Manager environment. This includes showing the following activities:

Activities in IBM Rational ClearCase:

- “Setting up IBM Rational ClearCase software repository” on page 11
- “Developing in Unified Change Management” on page 26
- “Build and Stage development artifacts” on page 32

Activities in IBM Tivoli Provisioning Manager:

- “Data Center modeling” on page 36
- “Workflow design and development” on page 55
- “Automation Package assembly and installation” on page 62

At a high level, Figure 2 on page 4 shows the sequence of activities demonstrated in our case study scenario environment.

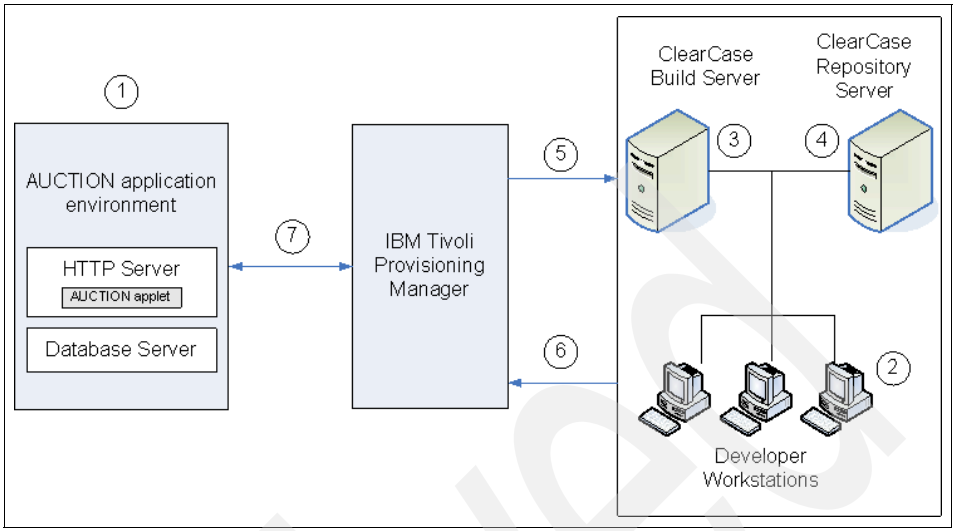

Figure 2 Auction application provisioning

1. The Auction application running on the production environment is experiencing several crashes and interruptions. Upon further investigation by the support team, it was found that the Auction application's main applet file itself was causing the problems.
2. The development team lead initiates the development process to address the reported errors. The relevant baseline in development is isolated to initiate the problem correction process.
3. Once the problems are fixed in development they are then integrated. The development team lead initiates the build process that includes the changes for the fix and places them in a baseline.
4. The build script stages the derived binary for the Auction application and places it in the designated location in the IBM Rational ClearCase repository and copies the binary to a software repository for IBM Tivoli Provisioning Manager.
5. The Support team is notified the new version of the Auction application that fixes the reported problems is available. The support team then uses the workflows in IBM Tivoli Provisioning Manager to initiate the deployment process and update to the relevant servers of the Auction application production environment with the latest version of the Auction application.
6. IBM Tivoli Provisioning Manager prepares to install software by checking for the server template associated with the application tier. The server template provides network, storage, and software associations via software stacks for the application tier to be provisioned on the selected server or servers. The

software stack contains one or more software module entries and may contain other software stacks as well. A software module will provide a software product definition which contains software installables definitions for the product. The software installable indicates the location of the file repository for the installable files. In our scenario the file repository is IBM Rational ClearCase repository. These definitions are presented in “Data Center modeling” on page 36.

7. Software installations are performed as required. The workflows issue operations that access the new build of the Auction application and places it into a staging area in the target servers, backs up the existing application files, and executes the installation process. This step requires the proper workflows be assigned to enable the logical operations to occur. For our scenario, these workflows are presented in “Workflow design and development” on page 55.

Existing IT infrastructure

To demonstrate the application provision scenario using IBM Rational ClearCase and IBM Tivoli Provisioning Manager for this scenario, the following assumptions have been made:

- A functional environment IBM Rational ClearCase SR5 (Service Release 5) environment that has Unified Change Management (UCM) enabled
- A functional environment for IBM Tivoli Provisioning Manager V3.1 is up and running with fix pack 1 applied.

The following figure shows the infrastructure used in this scenario. It serves as a starting point for us to create an environment that allow us to integrate demonstrate how IBM Tivoli Provisioning Manager can make use of a IBM Rational ClearCase repository.

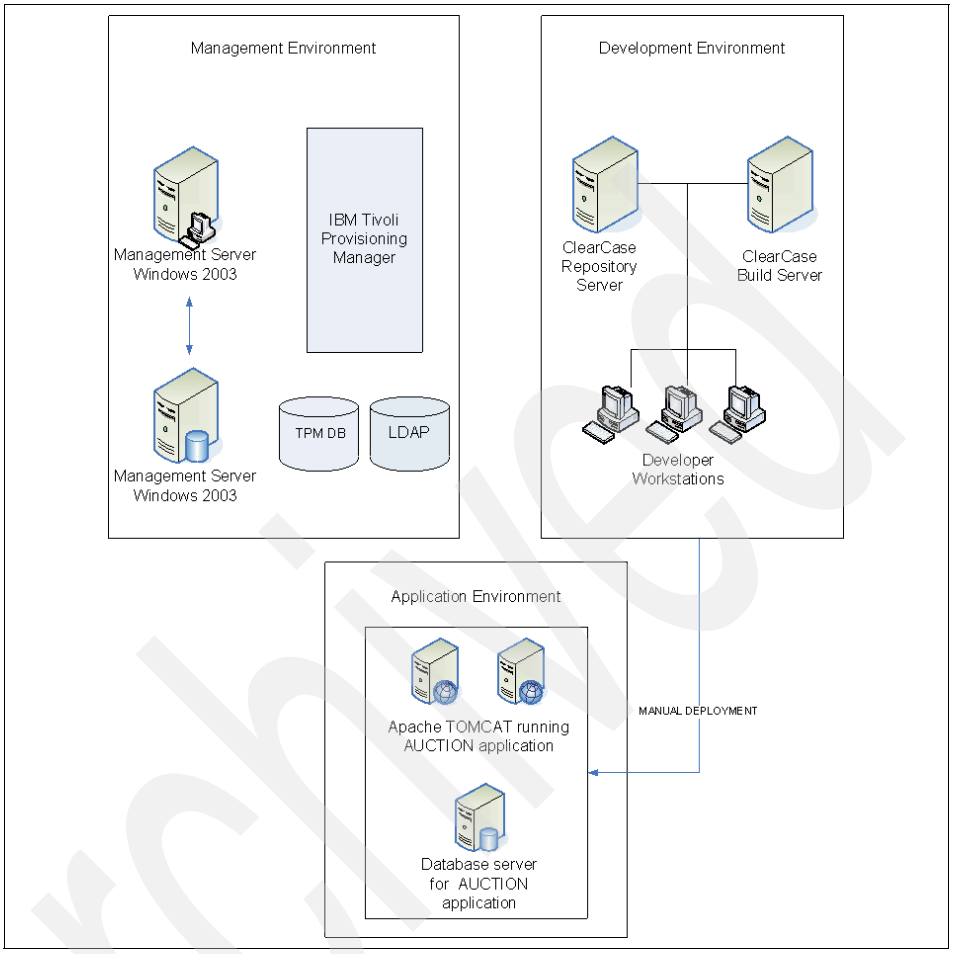

Figure 3 Pre-integration infrastructure

Development environment

In our scenario the development team is using IBM Rational ClearCase SR5 (Service Release 5) on a windows platform. The customer uses Apache Ant v1.6.5 for producing the application builds (`http://www.apache.org`).

The development team uses Unified Change Management (UCM) as a development process to do development. The UCM approach is a structured way of doing parallel development in a IBM Rational ClearCase environment. Some of the terms used in UCM are as follows:

Activity Tracks the changes you made to an artifact to address a change request

Component	A set of related artifacts that are developed, integrated and released together
Baseline	A named stable configuration that represents the integrated work of all the team members
UCM project	An object that contains configurations and policies relating to a development effort to create products, Group of products or a subset of a product's functionality
Streams	Are IBM Rational ClearCase objects that consist of baselines and activities and determine which artifacts should be made visible based on rules imposed by baselines and activities. There two types of streams development and integration. Development Stream allow developers to be isolated and work in a parallel development environment once a work has been completed in the development stream the developer would then deliver the work to an integration stream. The integration stream where changes from all developers are bought together and baselined
Views	Let a user select a set of versions of files and directories without having to specify the versions explicitly. In the UCM environment is selection is of files and directories is done by the streams and view becomes a means to access objects stored in ClearCase®.

Figure 4 shows the dependency between views, streams, baselines and activities.

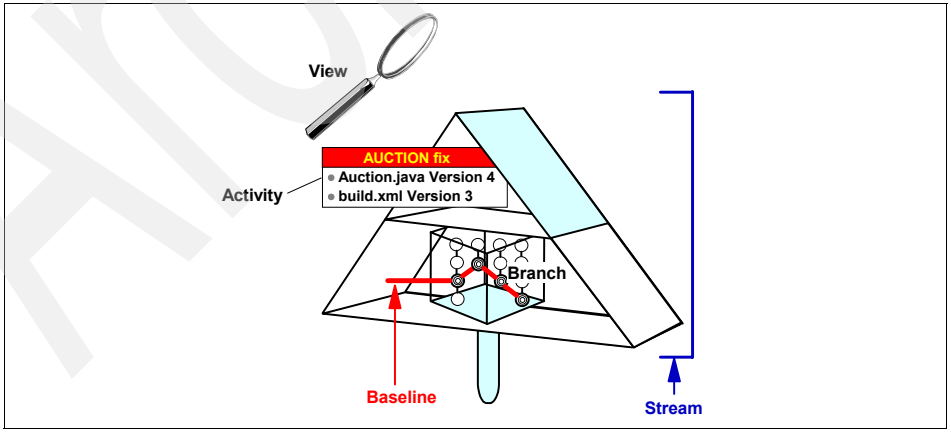

Figure 4 dependency between views, streams, baselines and activities

As seen in Figure 3 on page 6, the development environment used in this scenario consists of a dedicated IBM Rational ClearCase repository server and dedicated build server to do software builds. Both servers are running Microsoft® Windows® 2000 Server Edition and the developer workstation consist of Microsoft Windows 2000/XP professional edition. The build script for generating the Auction application installable files (Auction.jar in this case) file is written using Apache ANT as it provides a platform independent build environment that allows customers to build binaries for Java™ and .NET based applications.

The developers will access IBM Rational ClearCase using the IBM Rational ClearCase client installed on their workstations. IBM Rational ClearCase integrates directly with the windows explorer and with other leading development environments such as Eclipse, IBM Rational® Application Developer and Visual Studio® .NET. In our scenario the Auction application is developed in Java (Auction.java) using Eclipse. The model that IBM Rational ClearCase supports is a typical check-out, revise and check-in. Every time a developer checks in code IBM Rational ClearCase creates a version within the repository.

IBM Rational ClearCase also supports build tools such as **`make`** but has a variant of make called **`ClearMake`** which does build auditing and binary sharing. Build auditing generates a bill-of-materials during the build process that lets the users know which version of source files went into building the binaries and binary sharing which means IBM Rational ClearCase would not rebuild binaries that have already been built using the same configuration and just reuse those ones that have already been built. For the ANT environment IBM Rational ClearCase does not do binary sharing but will support build auditing in its upcoming release. In our scenario we are not using build auditing for ANT.

The deployment process currently in our case scenario customer environment is being done manually using FTP and usually becomes error prone when trying to deploy the Auction application to multiple servers at once. Also, managing the configuration of each server to run the application is a challenge.

The process for deploying the customer's custom developed Auction application has to follow a series of steps prior to deployment. The process requires that the applications or patches before being deployed to production are first deployed in the test environment and then to pre-production and finally to production. Even the deployment to production has three phases: backup the current environment, deploy the new application or patch installables to a staging area on the production server, and finally installing the application only if both prior steps are performed successful.

One of the benefit our customer has is that they have IBM Tivoli Provisioning Manager in place to manage some of there other packaged software such as Windows workstation software. Using this capability of IBM Tivoli Provisioning Manager, in this paper we show how to implement a process of deploying

custom developed applications from IBM Rational ClearCase to the destination environments using IBM Tivoli Provisioning Manager.

IBM Tivoli Provisioning Manager environment

Our scenario is using IBM Tivoli Provisioning Manager in a two server deployment configuration. In this environment one server acts as the provisioning server, running IBM Tivoli Provisioning Manager Version 3.1 with fix pack 1 installed, IBM WebSphere Application Server V5.1, with fixpack1 and cumulative fix pack 3, IBM DB2® Universal Database™ Enterprise Server Edition Version 8.2 with fix pack 7 and hot fix 13832, and IBM Tivoli® Directory Server Client Version 5.2.

The second server acts as the directory server node running IBM DB2 Universal Database Enterprise Server Edition Version 8.2 with fix pack 7 and hot fix 13832, as well as IBM Tivoli Directory Server Version 5.2 server and client.

For deployments in which the services for the IBM Tivoli Provisioning Manager environment reside on two machines, the minimum hardware requirements are listed below.

- 2.8 GHz Intel® Pentium® 4 processor or equivalent
- Minimum of 4 GB of free RAM
- 20 GB of disk space

As our installation has been implemented on Microsoft Windows 2003 Server operating system, there are some additional requirements to make the management environment function correctly such as Cygwin to enable secure communication between the management server and managed servers and devices, and IBM Tivoli Provisioning Manager fix pack 1 to provide updates to IBM Tivoli Provisioning Manager and existing automation packages required for this scenario.

As we stated earlier, it is assumed the IBM Tivoli Provisioning Manager has been deployed and has a functioning Data Center Model (DCM) configured. Later in this paper we will describe in detail the steps required to add the entire application infrastructure to the existing DCM, including defining the customer, application(s), cluster(s), software definitions for products, dedicated servers, and configuration templates for servers and software definitions. We will also include the definition of the IBM Rational ClearCase software repository to support the provisioning process in IBM Tivoli Provisioning Manager.

For additional information about IBM Tivoli Provisioning Manager, refer to the IBM Tivoli Provisioning Manager V3.1 Info Center at:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.tivoli.tio.doc/tio_nav.xml

Scenario implementation overview

The goal of this paper is to implement an environment similar to Figure 5.

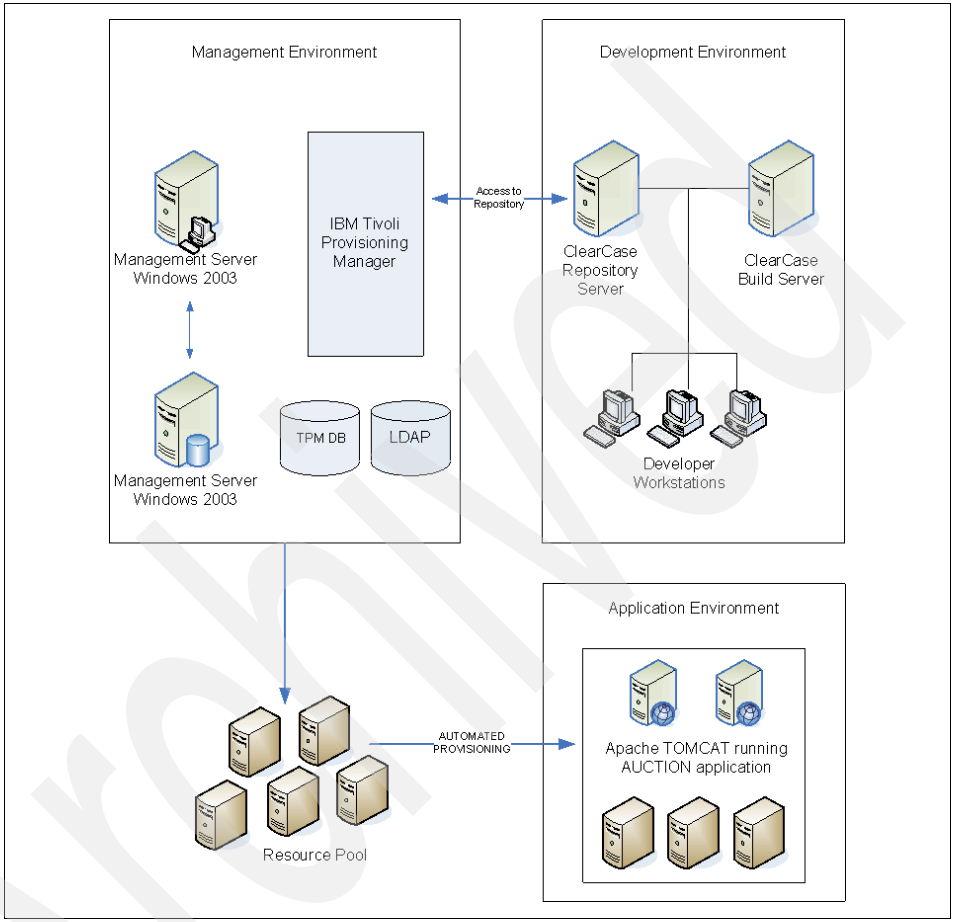

Figure 5 Integrated environment infrastructure

In order to achieve an integrated development to deploy model using IBM Rational ClearCase and IBM Tivoli Provisioning Manager, the following need to completed:

- The development environment has been configured properly. It includes setting up the IBM Rational ClearCase build and repository servers, and all IBM Rational ClearCase clients. Also Ant v1.6.5 and Java Development Kit 1.4 or higher installed on the development workstations and IBM Rational ClearCase build server.

- Auction application files have been loaded into the IBM Rational ClearCase environment and the clients can access the IBM Rational ClearCase repositories
- The latest level of all required prerequisite automation packages need to be installed and verified in the IBM Tivoli Provisioning Manager environment. Our case study scenario has the Apache automation package as pre-requisite.
- The Auction application environment needs to be completely integrated into the existing IBM Tivoli Provisioning Manager Data Center Model.
- Workflows to carry on the provisioning operations of our Auction application cluster must be developed and tested using any of the IBM Tivoli Provisioning Manager provided workflows development environments: Workflow Composer and the Automation Package Development Environment (APDE).
- Create an automation package that includes all workflows, device drivers, scripts, configuration templates for server and software definitions, our sample application installable module, and other DCM sample definitions. This automation package must be installed in our IBM Tivoli Provisioning Manager environment.

The following sections detail each of the tasks described above.

Setting up IBM Rational ClearCase software repository

In a IBM Rational ClearCase environment the repositories are referred to as a Versioned Object Base's (VOB's). To achieve our integration environment using Unified Change Management we need a IBM Rational ClearCase Project VOB (PVOB) and a VOB. A PVOB is a special type of VOB that is used when UCM is implemented as the software configuration management process. Every UCM project belongs to a PVOB and multiple UCM projects can share a PVOB.

The process to create VOBs can either be done through the command line or from a graphical user interface (GUI) based wizard. The Graphical interface is invoked from **Start** → **Programs** → **Rational Software** → **Rational ClearCase** → **Administration** → **Create VOB.** This start the wizard as shown in the following figure.

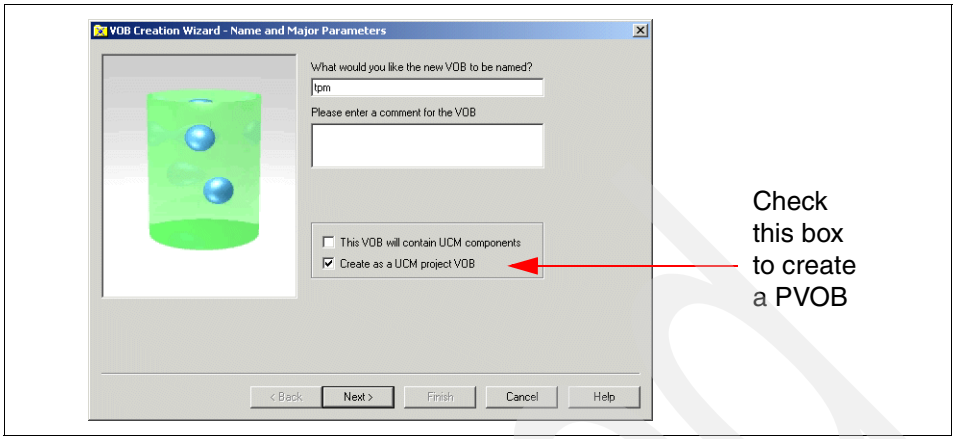

Figure 6 Creating a PVOB - tpm VOB

Public VOB's allow all users to mount the VOB on a workstation (this is especially important when you are using dynamic views). A private VOB has to be explicitly mounted only by its owner. In our case study scenario, we chose to create a public VOB as all developers can mount the VOB as necessary.

To enable public VOB creation set the IBM Rational ClearCase registry password first using C:\Program Files\Rational\ClearCase\bin\rgy_passwd. Once the registry password is set you can use the same password to create public VOB.

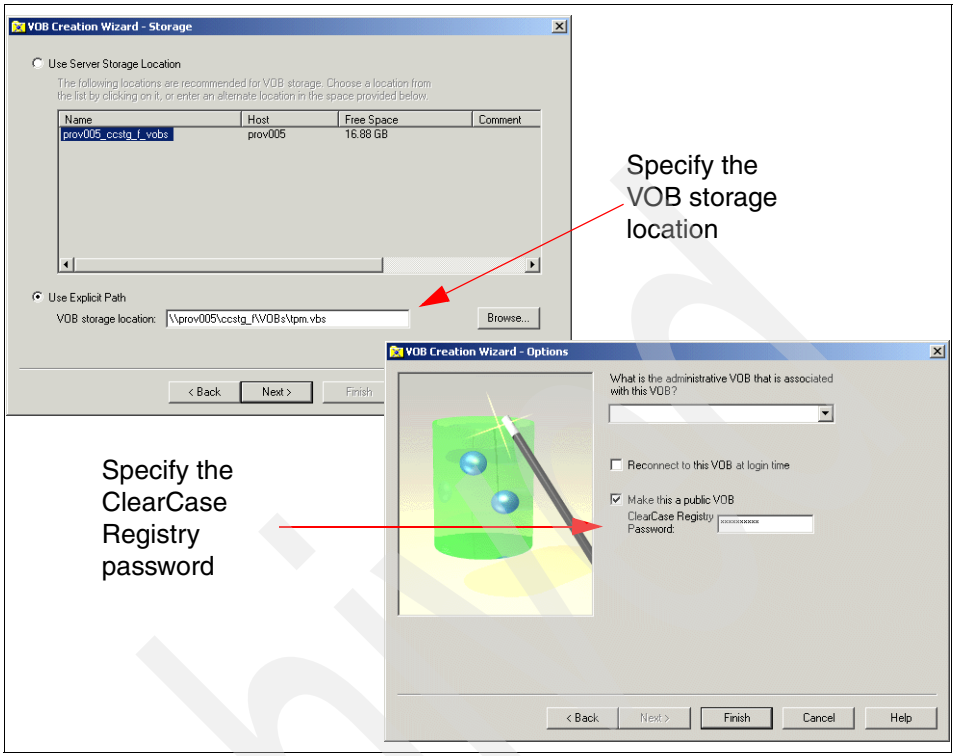

Figure 7 Creating a PVOB - tpm VOB

The command line equivalent of the Wizard is shown below.

```
cleartool mkvob -ucmproject -tag \tpm -public <clearcase password> <repository location>
```

Where `<repository location>` is the location where the repository will physically reside on the repository server. In our case study environment, it is set to \\prov005\ccstg_f\VOBs\tpm.vbs.

To create VOB that will hold source and binary information for the Auction application we perform the following tasks:

- Create a Base ClearCase view
- Create the VOB that will hold Auction UCM components

The process to create a Base ClearCase view is needed since we are using a component boundary called sub-VOB components. Sub-VOB components behave like general VOB level components but have a small boundary than their VOB level counterparts. For details on sub-VOB, refer to the redbook *Software*

Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM, SG24-6399.

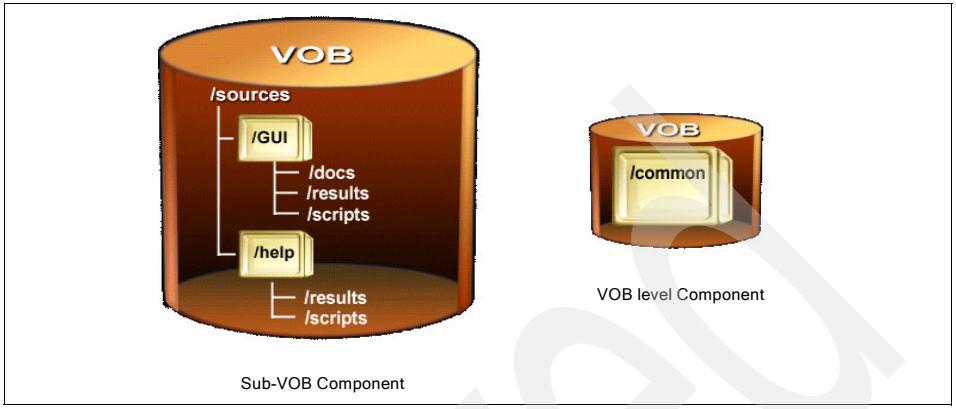

Figure 8 Sub-VOB and VOB components

In order to create a Base ClearCase view use the following command:

```
cleartool mkview -tag my_view  <view storage location>
```

Where `<view storage location>` is the physical location of the view. In our environment we set the view storage location to \\prov005\ccstg_f\views\my_view.vws.

In order to create the VOB that will hold Auction UCM components use the Create VOB wizard as shown in the following figure.

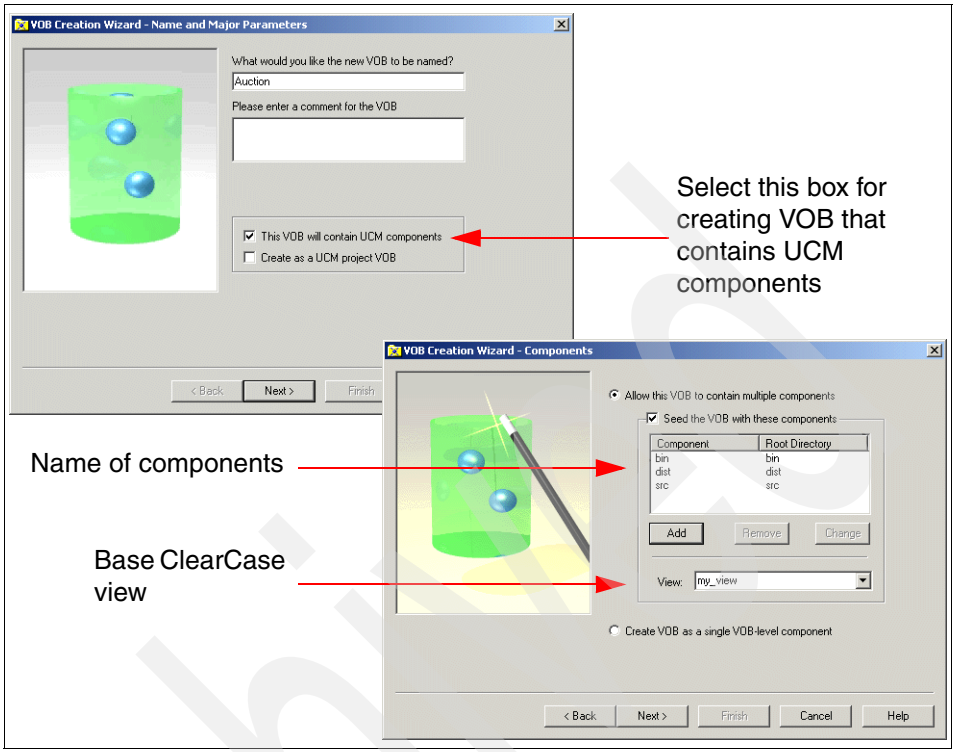

Figure 9 Auction VOB

The Auction VOB contains three components in the Unified Change Management environment which are as follows:

Bin The component that holds the binaries

Dist The component that versions the archives that deployed to the relevant environments

Src The component that holds the source code

At the end of the Create VOB creation wizard make sure that the field that asks for the project VOB association has the project VOB that we created earlier (tpm VOB). The Auction VOB must also be public VOB just like the tpm PVOB, as seen in the following figure.

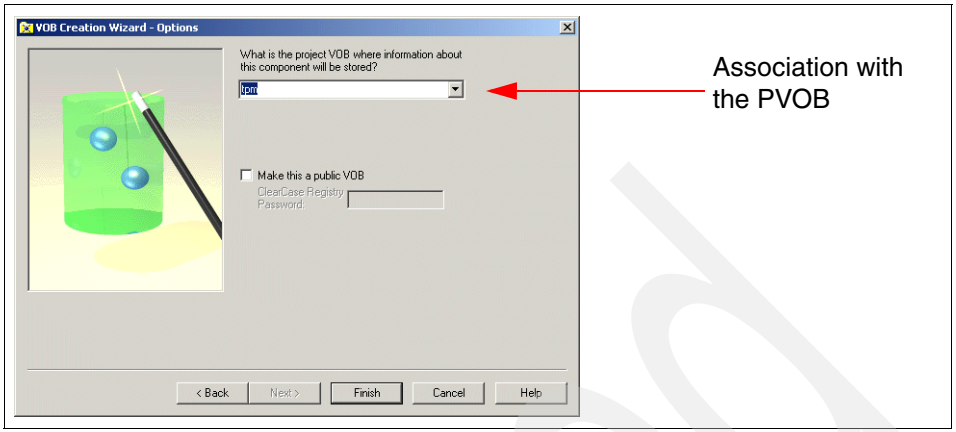

Figure 10 Association with the PVOB

Once this step is completed we will organize the project in UCM and load the base information for development to happen.

Organizing the project

After the both VOBs are created, we now organize the UCM project using the ClearCase project explorer. The UCM project explorer is the interface which allows the user to manage the policies that help guide the development effort in a structured way. To Start the project explorer you go to **Start** → **Programs** → **Rational Software** → **Rational ClearCase** → **Project Explorer**. The project explorer is a useful interface for highlighting the relationships between projects, stream, baselines and activities. The following steps need to be done to create a UCM project using the project explorer.

Select the **File** → **New** → **Project** operation to create a new project which will start the create project wizard.

The Create Project prompts for a project name and provides a choice whether the development project must be traditional parallel development or a single stream development. For our scenario we will select the Traditional Parallel development. The project name we use for our scenario is Auction_Rel1.

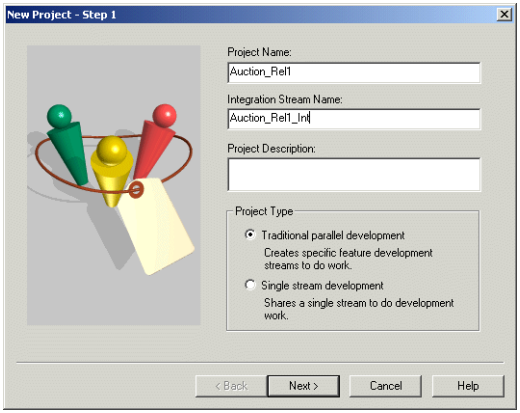

Figure 11 Auction project creation

The next step is select the components that will participate in the project with the respective baseline that will be the foundation for the initial work. Using the Add Baseline dialog box browser select the component and the press the change button and select All streams. Select the Initial baseline and proceed to do same step for the other two components.

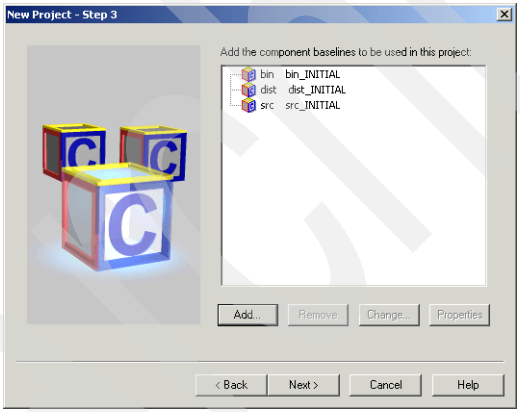

Figure 12 Selecting Components for development in the UCM project

Select the behavior of the component in that project (i.e., modifiable or non-modifiable). In our scenario all the three components are modifiable. For our scenario we change the default recommended promotion for a baseline to built.

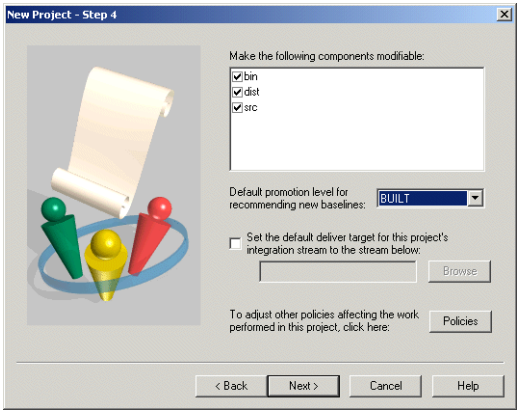

Figure 13 Final steps on creating the UCM project

We have now configured the project for our customer scenario, the next step is load our base data into our UCM environment.

Once the project gets created the project explorer now shows that the project which we created automatically gets the integration stream. The integration stream is where our customer will doing integration and product build to generate the Auction application installable files.

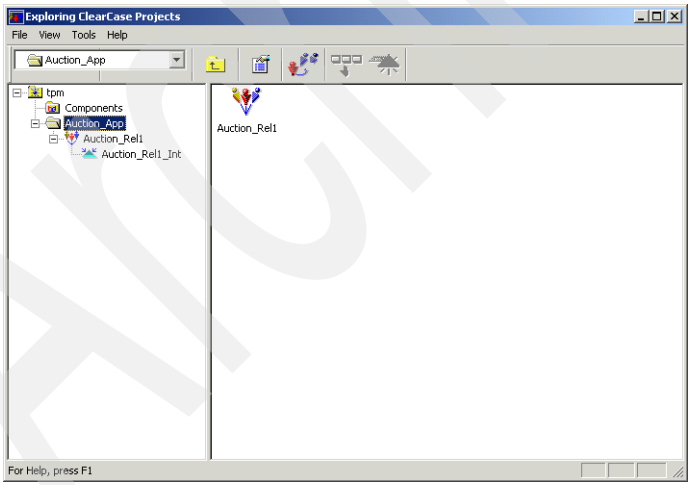

Figure 14 Project explorer after the UCM project creation

Loading initial application data into the repository

In our scenario, the initial Auction application data is provided by the Auction.zip file. To load the IBM Rational ClearCase repository with the initial data we proceed with the following steps:

- Extract the Auction.zip file to the C:\ Drive of the IBM Rational ClearCase VOB Server into a directory called Auction
- Create a view on the integration stream of the UCM project
- Copy the files from the extracted location (i.e., C:\Auction) to the Auction VOB
- Baseline the base artifacts and recommend it for the development teams

The Auction.zip file contains a directory structure that is similar to the Auction VOB created earlier in “Setting up IBM Rational ClearCase software repository” on page 11. The SRC directory stores the source files i.e., the *.java and *.xml files. In our scenario we will put the developed source code in this directory. The DIST directory will store the packaged jar file which will be copied to the Staging area for IBM Tivoli Provisioning Manager to deploy from, and the BIN directory contains the *.class files that results of build requests.

Now we will create the integration view using the project explorer and the view creation wizard. Going back to the project explorer expanding the Auction UCM project. Select the integration stream for the Auction project and then go to the **File Menu** → **New** → **View**. Selecting the option will start the view creation wizard for the integration stream.

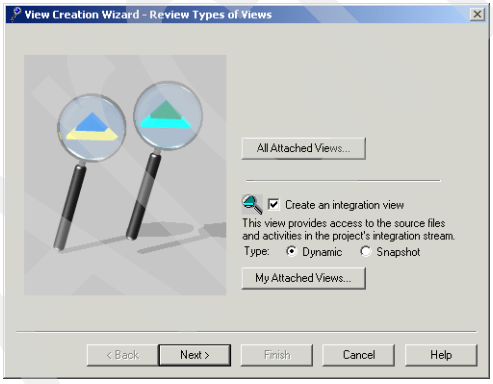

Figure 15 View creation wizard

In the view creation wizard you are given a choice to use either snapshot or dynamic views.

Dynamic view uses the ClearCase Multi-Version File System (MVFS) to provide immediate, transparent access to files and

directories stored in repositories. ClearCase maps a dynamic view to a drive letter in Windows Explorer.

Snapshot view copies files and directories from the repository to a local directory on your computer.

In our scenario we use a dynamic view. Once the dynamic view gets created, IBM Rational ClearCase automatically maps the view to a network drive (i.e., similar to Z: on windows) you can view the drive mapping using the windows explorer

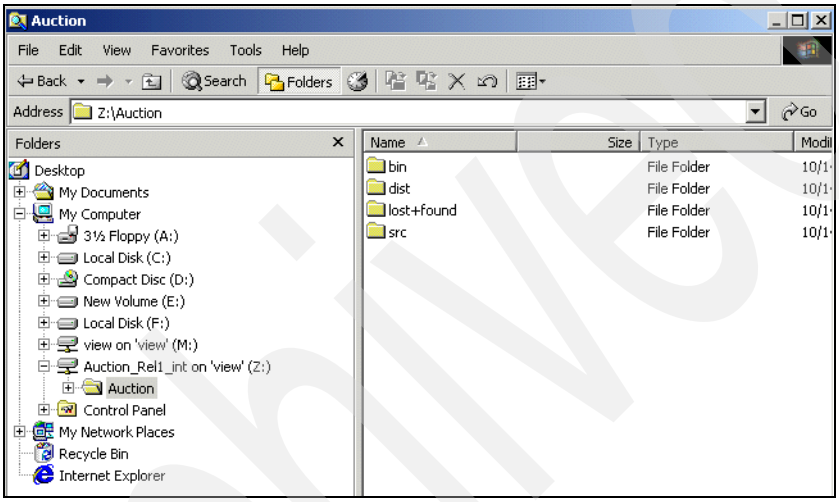

Figure 16 ClearCase windows Explorer drive letter mapping

Now that the view is created we can copy our files into the relevant directory of the IBM Rational ClearCase repository. As you can see from theFigure 16 the directories SRC, DIST, and BIN are already created as part of our Auction VOB creation process. In our scenario since it is a simple scenario we are going to copy and add the files to source control using the graphical user interface but in large enterprise scenarios it is recommend to use a command line tool called **`clearfsimport`**. This command is a single-pass converter, that is, elements are read out of the file system and placed into a VOB by the same program.

Since in our scenario we are going to add the files to source control by simply copying from the c:\auction directory placing them into the right directory inside the Auction VOB. Once that is done will use IBM Rational ClearCase functions to add them to source control, in this case the following happens:

- Auction.java and build.xml go into the SRC directory of Auction VOB
- Auction.jar goes into the DIST directory of Auction VOB
- Auction.class goes into the BIN directory of Auction VOB

Adding application files to source control

To add the files to source control, you can use the ClearCase explorer interface. Using the ClearCase explorer navigate to the view and into the relevant directory, highlight the files you want to add then select the **Tools** menu and choose add to source control.

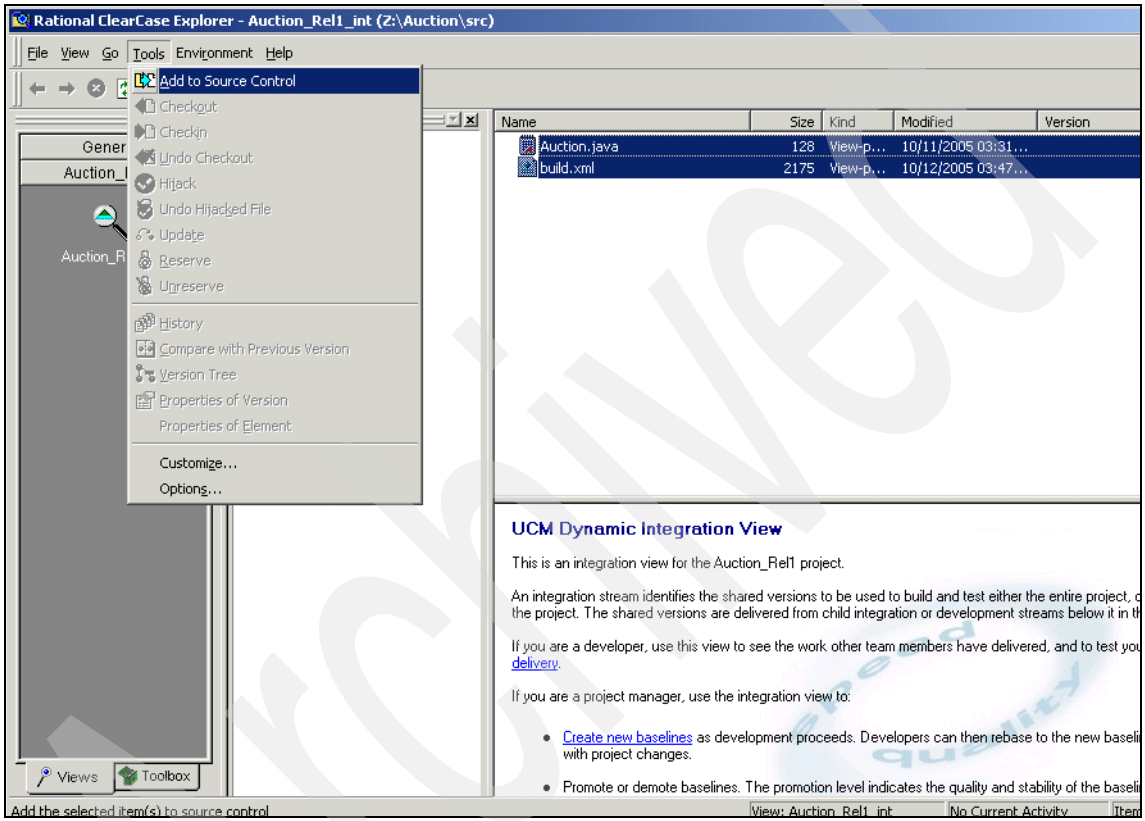

Figure 17 Adding files to source control

This has to be done for all the directories i.e., for DIST and SRC (hence using the **`clearfsimport`** command is better). While Adding the files to source control IBM Rational ClearCase will prompt you for an activity name, this is due UCM which expects that a user define the context of work they plan to do before modifying elements stored in the VOB. Since we are adding multiple elements make sure press the Apply to all button otherwise dialog will come every time you check-in a version of a file.

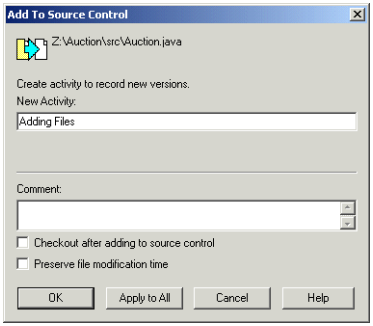

Figure 18 Activity dialog panel

Recommending baseline to development

Once the base data has been added using an activity we now recommend it as baseline for the development team to perform development. To do this, we use the ClearCase Project explorer to create a new baseline.

In the ClearCase project explorer we navigate to the integration stream of our Auction project and see what activities have been done on the stream. We should verify that the activity we used to add the files is listed on our integration stream view. The activities in their properties contain the version information of the files that were created or modified to address those activities.

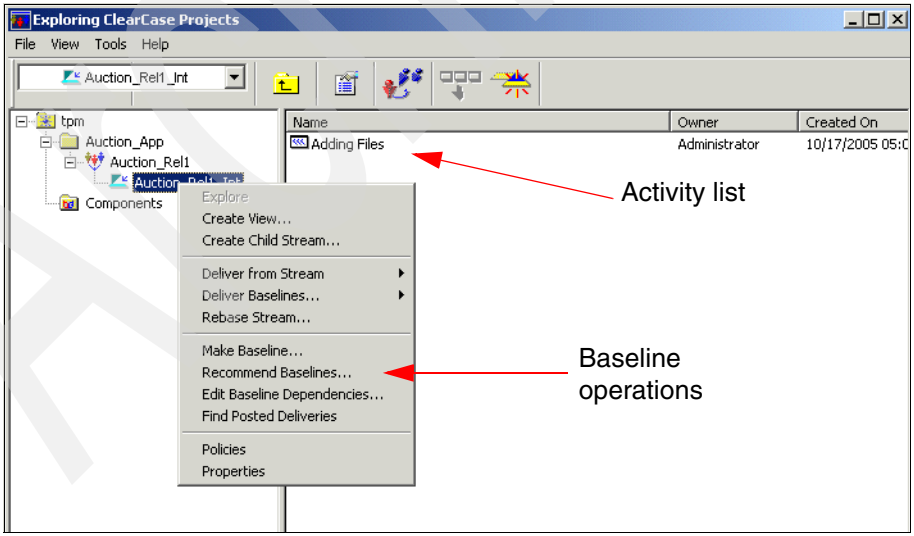

Figure 19 ClearCase Project Explorer - making a baseline

Once we are satisfied with the activities, we right-click the integration stream and select the make baseline operation. This will start the Make Baseline wizard.

The Make Baseline Wizard exposes tabbed interface, the first tab called General, contains basic information about what the baseline name would be (the default is the UCM project name followed by date and time stamp). In our case we have the name for the baseline as UCM project name followed by the Baseline number which in our case is Auction_Rel1_BL1. The other important information on the General tab is the kind of baseline i.e., whether baseline is incremental or Full. Full baseline labels all the versions of a component, even those that have not changed. An incremental baseline labels only versions that have changed since the last full baseline was created. We can use incremental baselines if we have a daily build strategy and a full baseline for a weekly build strategy. In our scenario we are creating a Full baseline.

On the second tab called Activities, the version of the files and directories that are part of an activity can be selected i.e., activities that need to be included in a baseline can now be selected and put into a baseline. Activities that are not selected will not be included in the baseline which means that the changes bought by those activities may not be exposed to the development teams. Once the activities are selected we click the OK button, which then proceeds to create the baselines for the relevant components, as shown in the following figure.

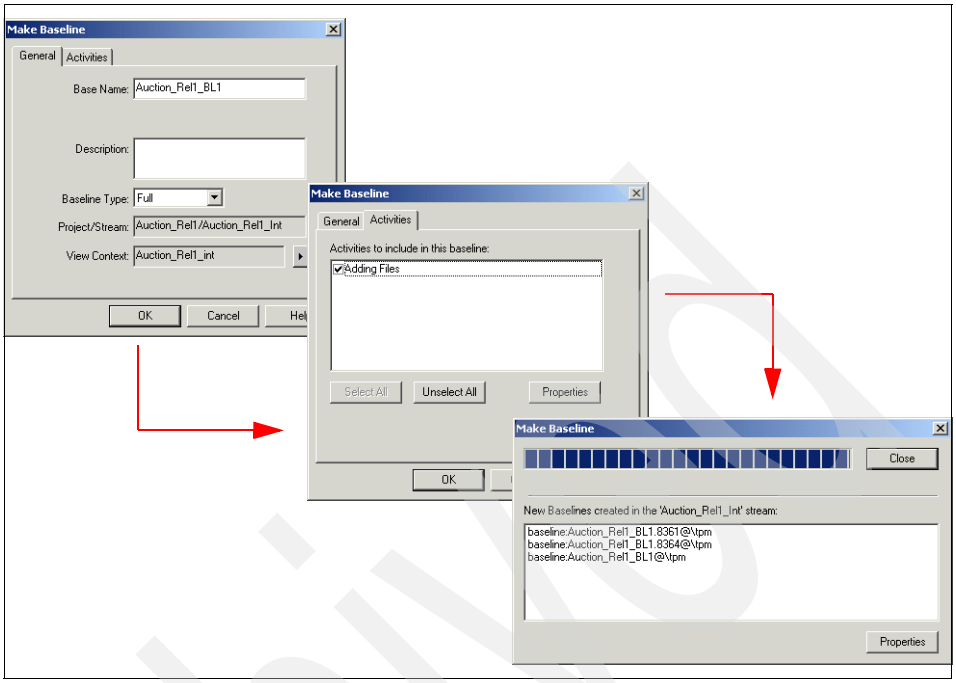

Figure 20 Make baseline wizard

After the baselines are created they need to recommended so that development teams can work of a recommended baseline. In order to proceed with that process, we go back to the project explorer interface and select the recommend baseline operation.

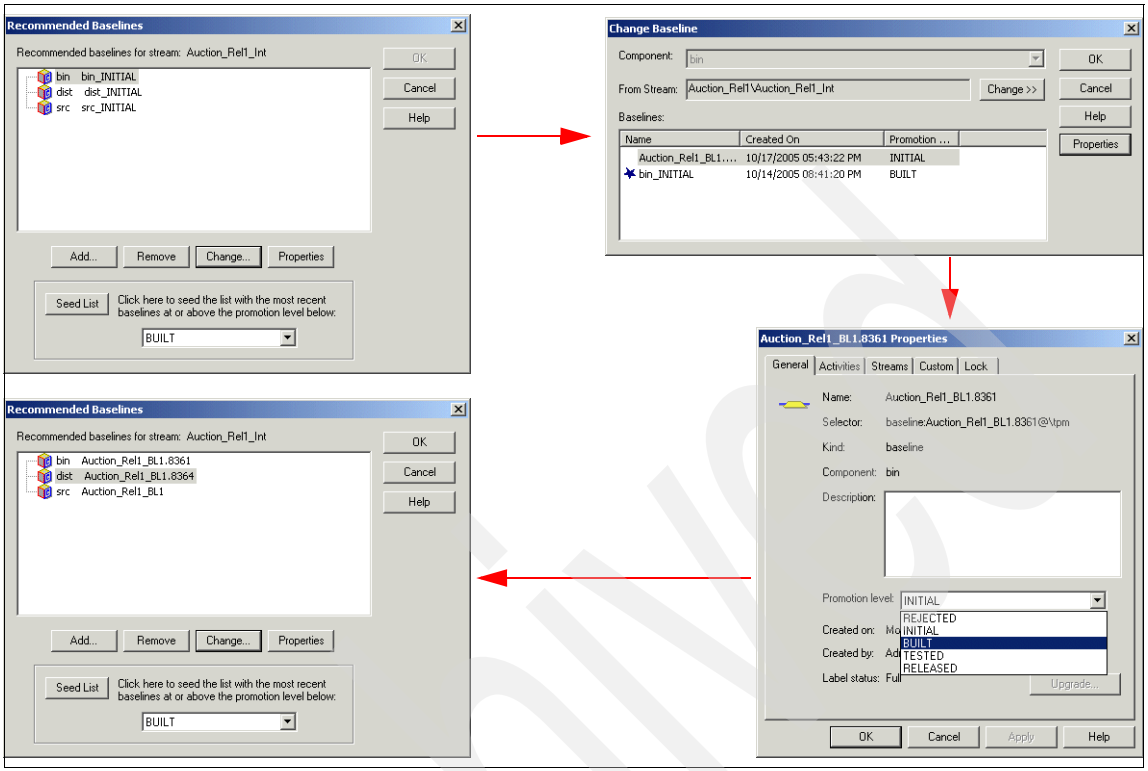

Figure 21 Recommending baseline and changing promotion level

To recommend a baseline requires a few steps to be undertaken. Every baseline that we create in a UCM project by default inherits the INITIAL promotion level. A promotion level signifies the goodness of a particular baseline for development to proceed to do work on. A recommended baseline does not mean that is the latest piece of work. What it means is that the most stable version of your code is now available for development to base their work from.

Once we open the recommend baseline wizard we want to change the baseline that we are currently recommending for bin, src and dist. To change the baseline press the change button on the wizard and the change baseline/component selector is invoked. Using the change baseline interface we select the relevant component, highlight the baseline we want to recommend, and select properties to change the value from INITIAL to BUILT. We will do this for all the other components. We can mitigate the additional steps for all the components if we composite baselines. Once all the right baselines have been selected and the promotion levels changed, we click the OK button back on the recommend baseline wizard. The new Auction baseline will now be the foundation of our scenario execution presented later in this paper.

Developing in Unified Change Management

As a developer using IBM Rational ClearCase UCM, you must understand what project or projects you are working on, where your IBM Rational ClearCase views are located, and what activity or activities you are performing.

The first step is to establish your working environment. IBM Rational ClearCase does this automatically when you join a project. The second step is to make changes accomplish a specific activity or task. This is done by checking out elements, making changes (editing files), and checking the elements back in. As elements are checked out and in, new versions are created in the VOB and associated with the activity you are working on. These versions are referred to as an activities change set.

Once you have finish making changes and testing those changes, you deliver your changes to your projects integrator. This is done by performing a deliver operation and specifying the activity or activities you want to deliver. The versions that get delivered are dictated by the activity or activities you select using the activity's change set. On a periodic basis you update your workspace (i.e., rebase) with changes that have been made by other developers on the same project's integration stream. Once the assets have been baselined and recommended the development team can start doing work on development activities, as shown in the following figure.

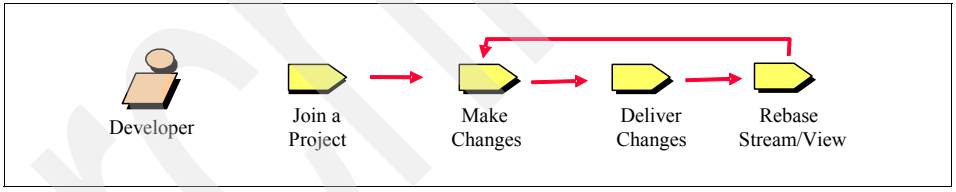

Figure 22 Development activities

Join a Project

In a UCM environment developers work in a UCM Project environment. A UCM project identifies a development work that delivers the following:

- A Product
- A Group of Products
- A Sub-System of a Product

The process of joining a project creates the following for a developer in a UCM project that supports parallel development:

- A Development stream specific to the developer
- Two views

– One view looking at the integration stream
– One view looking at the development stream

The integration view allows the developer to see changes that have been delivered by other users to the team integration stream and the developer view allows the developer to isolate their changes until the developer is ready to deliver their respective changes to the integration stream.

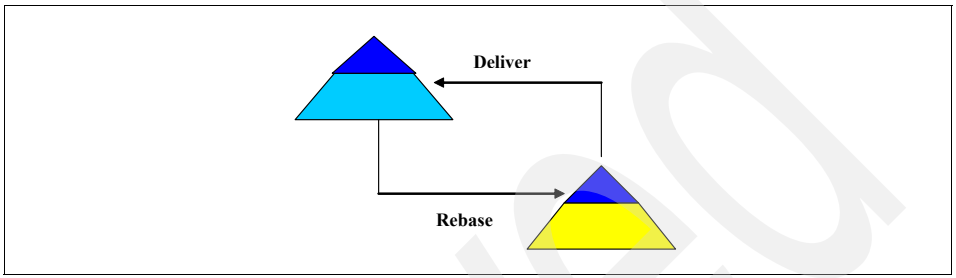

Figure 23 Relationship between integration and development stream

To Join the Auction project use the Project explorer interface and right-click the Auction project and select Join project that will invoke the Join Project wizard. During the Join project wizard it will prompt us to select the name of the development for the work we want to do as a developer and the type of views we want for our scenario. We will select the dynamic views for both integration and development stream. After specifying the type of views we will specify the view location on the workstation and select finish. This will create the views for the us to start our development process.

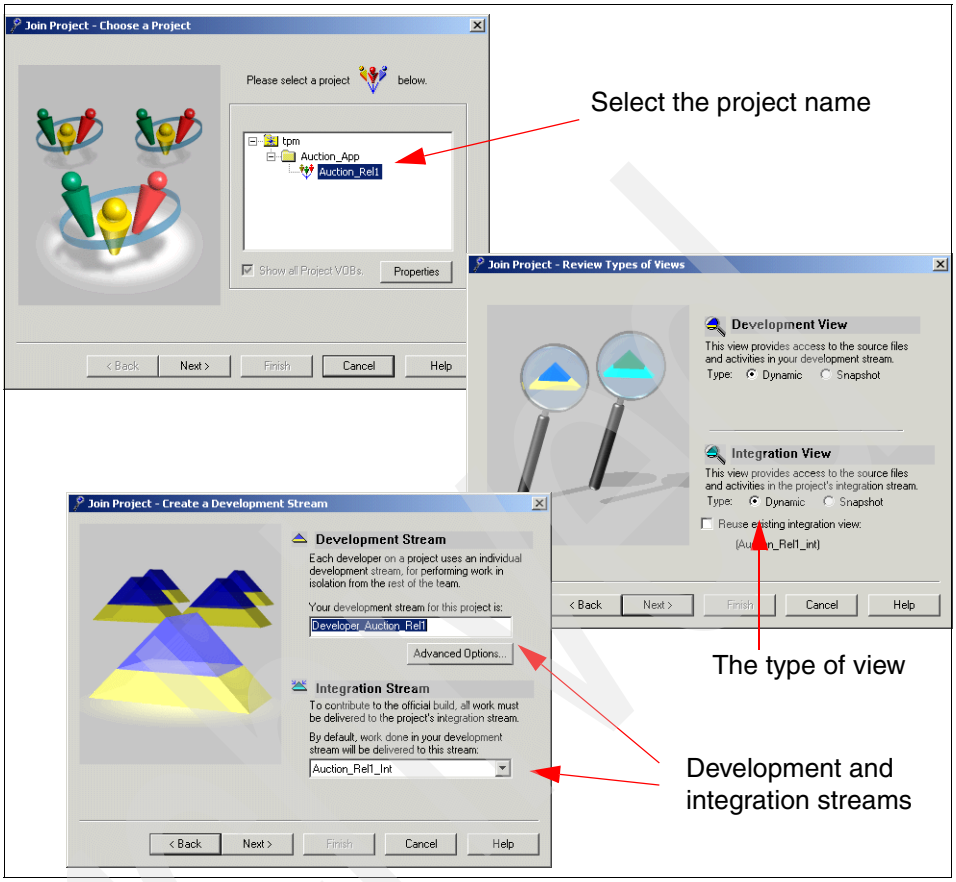

Figure 24 Joining an UCM project

Make Changes

After joining your project, you can now start making changes using either the ClearCase explorer interface or the windows explorer to check-in, check-out artifacts. Since the UCM model is activity based every time you check-out a file or a directory UCM will prompt you to enter an activity name (This is mandatory). Activities can map to requirements, Enhancement Requests or even Bug Fix. Using this paradigm it becomes easy for you to track to changes and progress on the respective tasks. Doing activity based development is also a value add on project management tasks as well, you can activities that map to project management tasks as well.

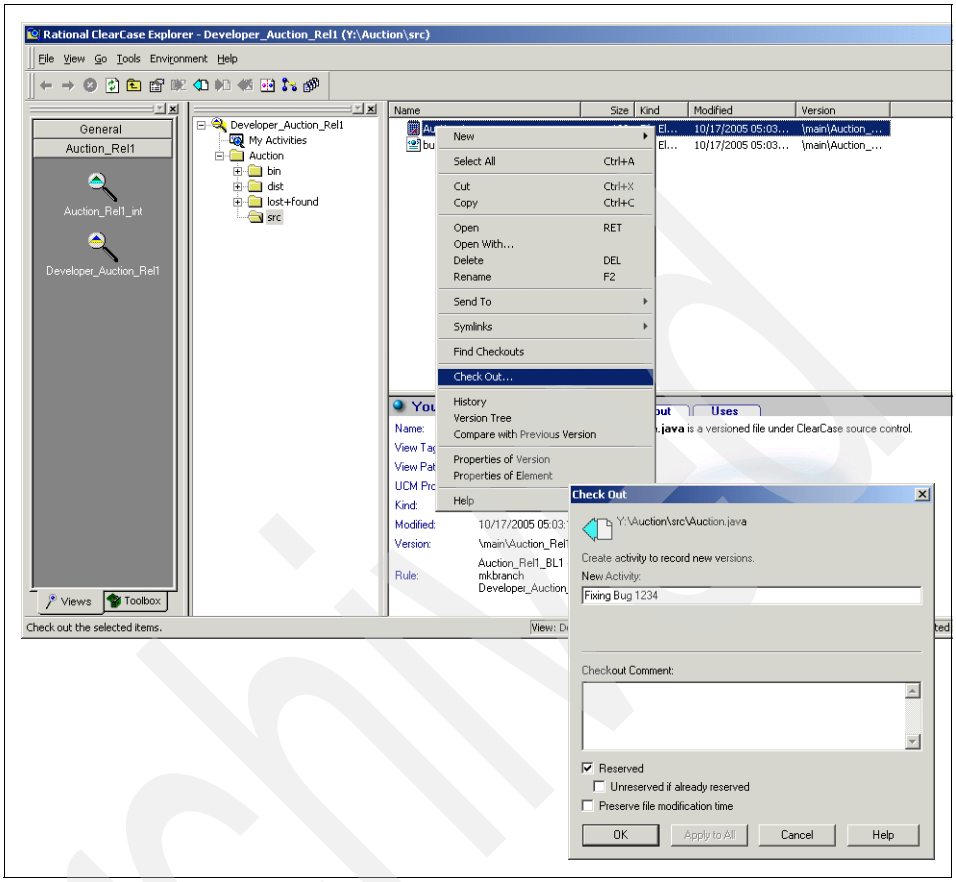

Figure 25 Checking out the source code to make corrections

In our scenario we are fixing Bug number 1234 which is causing the application to crash. Once the file is checked out the Auction.java is now modifiable, we can now go an edit the file and make our changes

Delivering the changes

At some point, you complete work on one or more activities. Because you are working in isolation, you need to take additional steps to make the changes you have made available to the project integration stream. The process of making your changes available to the integration stream is a called a delivery.

1. Delivering a change is multi step process:
2. Check in any outstanding checked-out elements.
3. Rebase from the project latest recommended baselines
4. Run the ClearCase deliver command.

5. Build and test the delivery (Unit test)
6. Complete or cancel the delivery

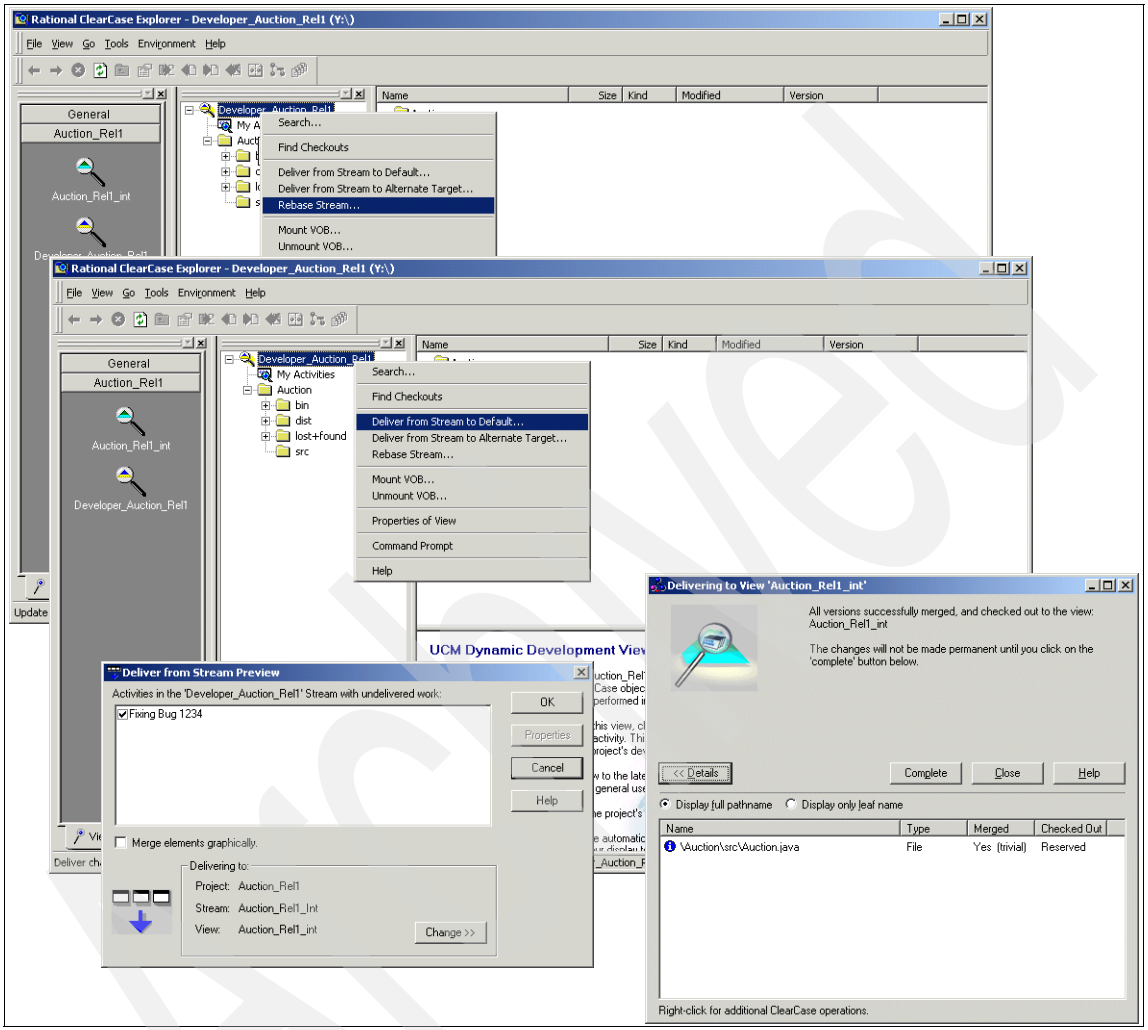

Figure 26 Delivering changes to the integration stream

As the shown in the figure above using the ClearCase Explorer in our development view we check in the changes we have made to the Auction.java file, proceeding from their we rebase our development view before we deliver our changes (this is done so as to not destabilize the working set in the integration stream with change made from the development stream and if the changes we introduce cause problems to the working set (i.e., baseline) on the integration, we will fix it in the integration stream. Implementing a rebase before deliver strategy helps in cutting down integration time.

After we rebase we start the deliver process, ClearCase prompt us to select which activity to we want to deliver, these can be on or more activities, in our scenario we are delivering the Fixing Bug 1234 activity. To complete the delivery we need to press the complete button to formalize the commit to the integration stream once the delivery is complete we can use the version tree browser to see what has transpired, as in the following figure.

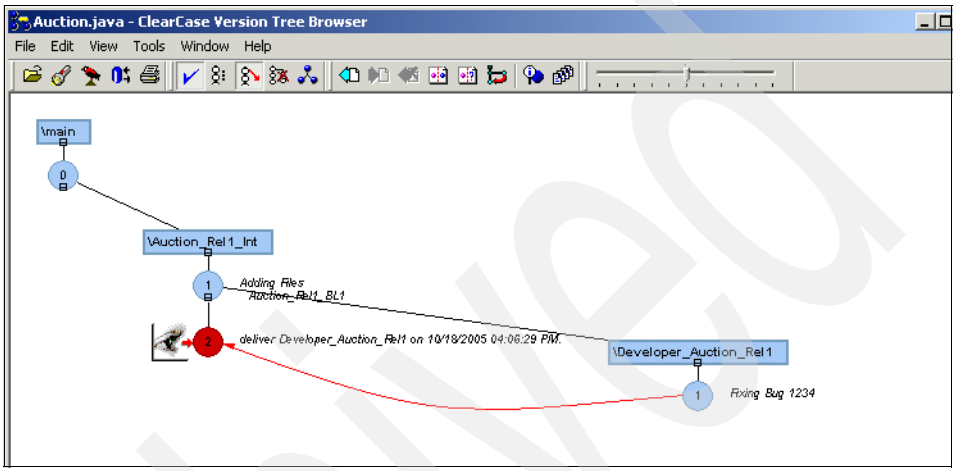

Figure 27 ClearCase Version tree after delivering changes to the integration stream

Rebasing the development stream

Our development view selects a stable set of element versions. We will always see the same versions in our development view unless you do something to change them. Periodically we need to update our development stream's configuration, thus updating the versions of the elements displayed in your view. This done by an action called Rebase.

Rebase updates our development view, making changes other developers have made visible to us. We do not need to get the latest and greatest element versions. Those versions may be broken or may not build together. Rather, we receive a stable set of baselines. The development lead creates new project baselines, builds, and tests them.

Once a baseline has reached a known level stability (this generally means passed some level of testing). The development team lead declares the baselines as the recommended baselines.

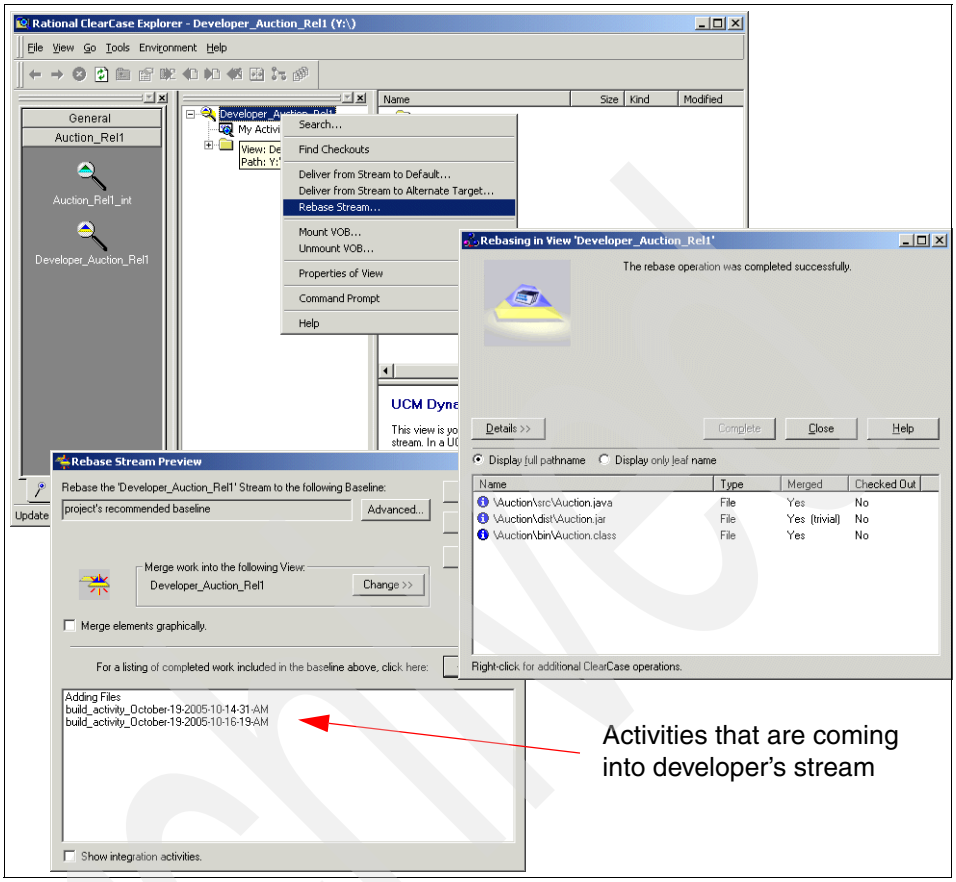

Figure 28 Rebasing activity

Build and Stage development artifacts

Integration is the process of bringing together independently developed changes to form a testable piece of software system. It can occur at many levels, eventually culminating in a complete software system. The larger the software system and the larger the teams working on that system, the more levels of integration are required to manage the software development efforts.

In our scenario the development team lead plays the role of an integrator. This person would validate all the deliveries made to the integration stream of the project and then run a build of the component and create a baseline.

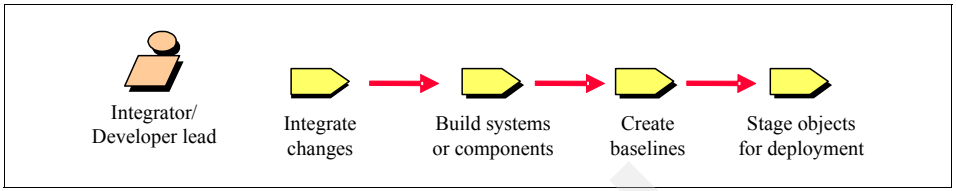

Figure 29 The build process

Once all the deliveries have been made, the integrator will then proceed to lock the integration stream. The project integrator will only have a view to the integration stream of the project. Locking the stream allows the integrator to stabilize code for the build and prevents late deliveries from the other users to the integration stream during the development process. The lock will only work on other users except for the person doing the build.

To setup a lock on the integration stream, using the UCM project explorer expand the Auction Rel1, right-click the Auction_Rel_int stream and select properties. On the Property Sheet select the Lock tab and click the Locked radio button. On the bottom half of the lock tab select exclude users section and right-click the exclude users section. Proceed to type in the name of the user ID that is going to do the build. In our scenario it is administrator ID.

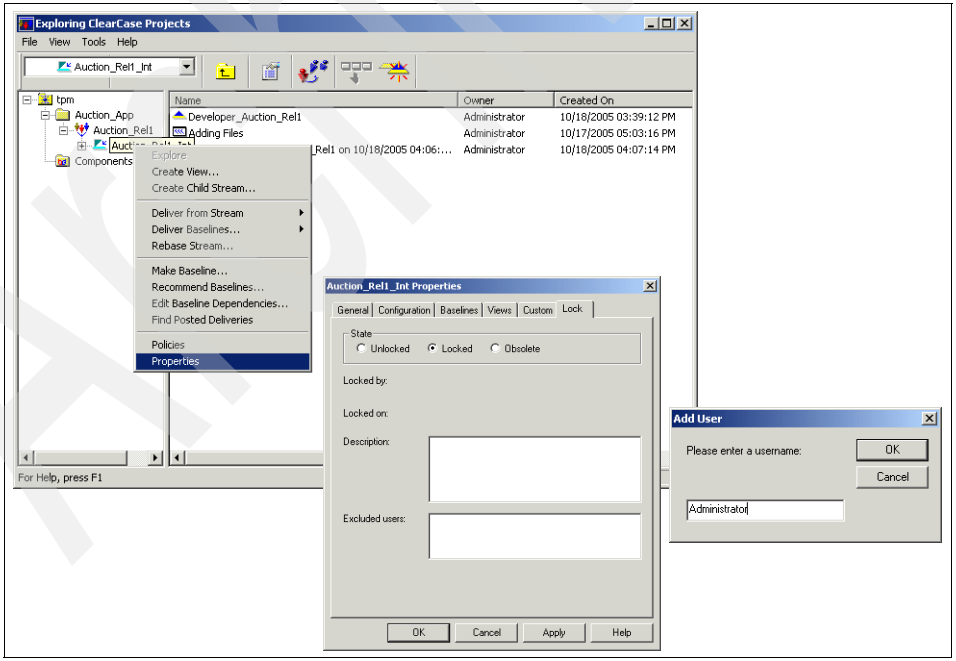

Figure 30 Locking the integration stream

After the integration stream is locked, we will now go ahead launch the command prompt and change directory into M:\Auction_Rel1_int\Auction\src. Within the src directory consist a build.xml file. The build.xml file is an input into ANT, which will read and generate the binaries, although ANT does have operations supporting IBM Rational ClearCase checkouts and check-ins.

For information sake in our scenario we just used direct <exec> calls to execute IBM Rational ClearCase commands. The ANT script to do the build is shown in the following example.

Example 1 Build ANT script for the Auction application

```
<?xml version="1.0"?>
<project name="Auction" default="main" basedir="../">
   <property name="src" location="src" />
   <property name="bin" location="bin" />
   <property name="dist" location="dist" />
   <property name="ClearCase_Stage" location="F:/ClearCase_Stage" />

   <tstamp>
      <format property="MY_TIME" pattern="MMMM-dd-yyyy-hh-mm-ss-aa"
locale="en_US" />
   </tstamp>
   <property name="bl_name" value="Auction-${MY_TIME}" />
   <!--=========================================================-->
   <!--      Build the code                                     -->
   <!--=========================================================-->
   <target name="build">
      <exec dir="." executable="cleartool">
         <arg line="setact -none" />
      </exec>
      <exec dir="." executable="cleartool">
         <arg line="mkact build_activity_${MY_TIME}" />
      </exec>
      <exec dir="." executable="cleartool">
         <arg line="co -nc ${bin}/Auction.class" />
      </exec>

      <javac srcdir="${src}" destdir="${bin}" />

      <exec dir="." executable="cleartool">
         <arg line="ci -nc ${bin}/Auction.class" />
      </exec>
   </target>

   <target name="jars" depends="build">
      <exec dir="." executable="cleartool">
```

```
            <arg line="co -nc ${dist}/Auction.jar" />
         </exec>

         <jar jarfile="${dist}/Auction.jar" basedir="${bin}">
            <manifest>
               <attribute name="User name" value="${user.name}" />

               <attribute name="Baseline" value="${bl_name}" />

               <attribute name="Build activity" value="${MY_TIME}" />
            </manifest>
         </jar>

         <exec dir="." executable="cleartool">
            <arg line="ci -nc ${dist}/Auction.jar" />
         </exec>
         <exec dir="." executable="cleartool">
            <arg line="setact -none" />
         </exec>
         <exec dir="." executable="cleartool">
            <arg line="mkbl -nc -all -full ${bl_name}" />
         </exec>

         <mkdir dir="${ClearCase_Stage}" />
         <mkdir dir="${ClearCase_Stage}/Auction" />

         <copy file="${dist}/Auction.jar" todir="${ClearCase_Stage}/Auction" />
      </target>

      <target name="clean">
         <delete dir="${ClearCase_Stage}" />
      </target>

      <target name="main" depends="jars" />
   </project>
```

The ANT script above performs the following operations

- Creates a build activity
- Checks out the binaries
- Compiles the source
- Generates a Jar file with the manifest information containing baseline name and build activity name
- Checks in the new changes
- Creates a new baseline
- Copies the JAR file to a staging location for Tivoli to deploy to a target environment

- If required you can ant clean and ant will delete the ClearCase Stage directory

Once the baseline is made, the development lead recommends the new baseline and changes the base line properties as shown in “Recommending baseline to development” on page 22. After the baseline is recommended the development team will rebase to the new baseline

As part of the build that was generated by ANT, one of the tasks was to copy the file to a source /staging area where IBM Tivoli Provisioning Manager could point to for transporting the source file. In our scenario the staging area is located on the F:\ClearCase_Stage directory. This staging area will serve as the software repository defined for our sample Auction application.

To make sure that the we can trace the binary to a development baseline, we can open the jar file and read the manifest file which will contain information as shown in following figure.

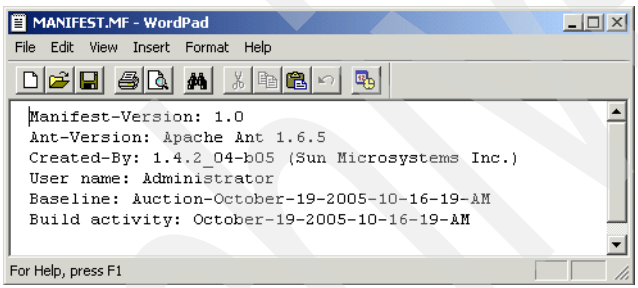

Figure 31 Manifest file for Auction.jar

At this point no further action is necessary in the IBM Rational ClearCase environment. The application development team has prepared a corrected version of the Auction application, which is available at a pre-determined staging area. Later in this paper, IBM Tivoli Provisioning Manager will be configured to capture the Auction application builds from the staging area and perform provisioning operations to the Auction production environment.

The following sections detail the tasks required to configure, for our case study scenario.

Data Center modeling

The Data Center Model (DCM) is a repository of all physical and logical resources that are managed by IBM Tivoli Provisioning Manager. Managed resources can be servers, switches, applications, customers, software and other

related equipment. It keeps track of changes made by workflows to the hardware and software configuration and keeps the model in sync with the real world hardware and software assets that are associated with it.

The DCM also stores information needed for the management of resource pools and clusters such as server ids, size of resource pools, number of active and inactive servers and server priorities, and the associated templates.

The DCM information is stored in a central database controlled by IBM Tivoli Provisioning Manager. It contains information regarding a real life data center. IBM Tivoli Provisioning Manager communicates directly with all components of the data center based on the definitions in the DCM.

DCM entries can be built using an XML file and imported into the DCM database using tools supplied with IBM Tivoli Provisioning Manager. To customize the existing DCM to enable IBM Tivoli Provisioning Manager to interact with IBM Rational ClearCase and provision the sample application used in this scenario, Auction application, we have to provide the following solution specific DCM definitions:

- Customer, application, and application tiers
- Resource pool and spare servers
- Software definitions, software installables, and software configuration templates
- Software stacks
- Server templates for both resource pool and application tiers
- Software repository

Note: In this paper, there will be many references to the terms *software definition* and *software module*. For our purposes, we are referring to the same IBM Tivoli Provisioning Manager object. If you use the IBM Tivoli Provisioning Manager GUI, you will notice the objects identified as software definitions. If you look in the Data Center Model XML file, you will see the objects referred to as software modules.

After the above list of objects is defined and integrated correctly into the Data Center Model, we use these definitions in support of the logical operations provided by an automation package to provision our sample Auction application.

Figure 32 on page 38 illustrates the relationships that exist between the above objects defined within the Data Center Model.

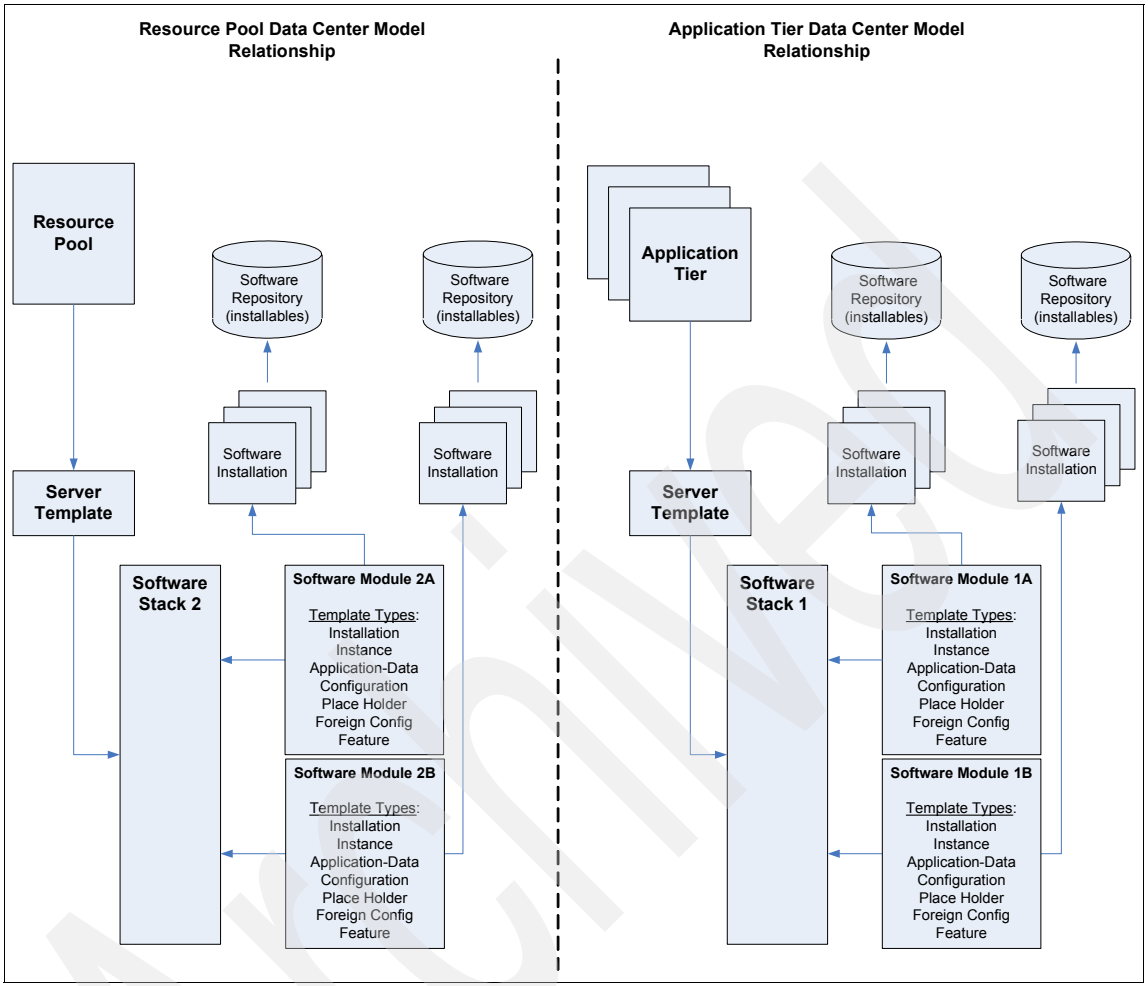

Figure 32 Data Center Model object relationships.

Based on Figure 5 on page 10, we need to create entities in the DCM that will represent our application topology.

According to the guidelines presented in the above picture, we need to define a DCM infrastructure that supports our application development and production environments. We start by defining a customer and an application. This application contains three tiers which will also be defined in the DCM. For the scenario presented in this paper we are only concerned with provisioning the application tier on which our sample Auction application runs: the HTTP Server tier.

We also define a server template, and associate that server template with the HTTP Server tier. The server template will identify the required software for the application. This is accomplished by defining a software stack, adding software modules to the stack, and associating the software stack to the server template.

In addition to software module definition for the Auction application itself, a software module definition for the HTTP Server on which the Auction application runs will be defined. Both the software module definitions, Auction and HTTP Server, will be assigned to the software stack definition of our application tier server template.

Each software module definition will have configuration templates which contain attributes used to install and configure the software. The software modules will also have an installation defined pointing to the location of the software installables. In the case of the Auction application, the location of its installables is the software repository of IBM Rational ClearCase.

In addition, we need to define a server template for the resource pool for our scenario. In the same fashion as we described above, the template is used to ensure the servers in this resource pool have the HTTP Server installed. This means they would require only the installation and configuration of Auction application when provisioned to our application tier.

The figure below illustrates the DCM elements and their relationships for the application tier definition of the sample Auction application.

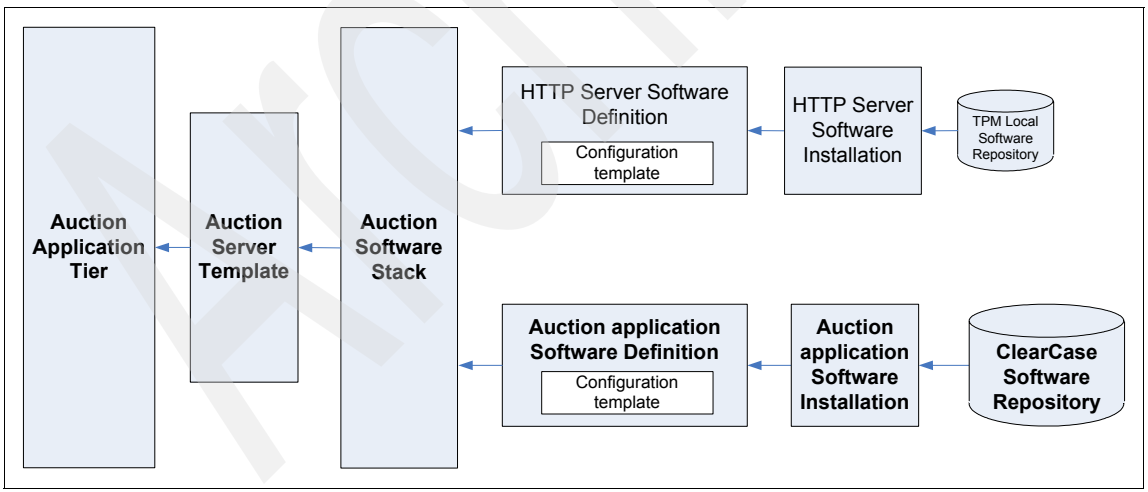

Figure 33 Auction application DCM object relationships

The following sections detail the DCM definitions for our case study scenario presented in the above figure.

Customers, Applications, and application tier definitions

IBM Tivoli Provisioning Manager requires certain objects defined to the DCM to enable management of the environment and to enable provisioning. The DCM maintains information about relationships between the different definitions once added to configurations or objects, so the proper information will be used whenever a provisioning operation is initiated.

Customers

In IBM Tivoli Provisioning Manager a customer is defined as an organization that is associated with one or more applications managed by your datacenter.

For our scenario, we performed a single customer definition in our data center, called `OnlineAuctionCustomer`. This customer definition is used to host the managed application and the three application tiers. These are described in the following sections.

Applications

In IBM Tivoli Provisioning Manager, applications consist of groups of servers which run the different levels of the application. For example, the three layers of the Auction application run HTTP, WEB application, and database services. The applications' priority can be set to help influence decisions concerning orchestration and provisioning of resources.

In our scenario we have defined one application, called `WEB Auction`. The application contains three tiers, discussed below.

Application Tiers

In IBM Tivoli Provisioning Manager, application tiers are groups of managed servers which run the separate tiers, or levels, of an application. An application tier can contain both dedicated servers assigned specifically to the tier, or overflow servers provisioned from a resource pool.

The application tier definitions we create for our scenario are used to integrate an existing application (Auction application) into our Data Center Model. In this case we define three application tier entries in the DCM to support each of the three tiers in our Auction application. These tiers definitions contain dedicated servers at the time they are created, as we are mirroring the pre-existing application infrastructure.

The application tiers for our case study scenario are:

- Auction WEB Servers, which will be the focus of our scenario
- Auction WEBAPPL Servers
- Auction DB Servers

Below we show the XML used for our Data Center Model customer, application, and application tier definitions. You will notice in the XML that the application tiers are referenced as clusters.

In the following example, the &win-saps entry is a pointer to the service access point definitions, which are in a different section of our XML file, for organizational purposes only.

As we are defining a pre-existing infrastructure for the Auction application, we also define dedicated servers for each of the application tiers. In the case of the Auction WEB Servers application tier, there are two dedicated servers: OAWebSrv01 and OAWebSrv02, as shown in the example below.

Example 2 Customer, application, and application tier definition for Auction application

```
<customer name="OnlineAuctionCustomer">
   <application name="WEB Auction" priority="1"  locale="en_US">
      <cluster name="Auction WEB Servers" min-servers="1" max-servers="9"
               pool="TOMCAT Pool" managed="true" is-device-model="Simulator"
               locale="en_US">
         <with-load-balancer name="Load Balancer 1"/>
         <server name="OAWebSrv01"  locale="en_US">
            <nic connected-to-switch="CISCO1" connected-to-module="fa0"
                 connected-to-port="24" management="true">
               <network-interface name="Local Area Connection"
                        ipaddress="9.3.5.159" netmask="255.255.255.0" />
            </nic>
         &win-saps;
         </server>
         <server name="OAWebSrv02"  locale="en_US">
            <nic connected-to-switch="CISCO1" connected-to-module="fa0"
                 connected-to-port="23" management="true">
               <network-interface name="Local Area Connection"
                        ipaddress="9.3.5.158" netmask="255.255.255.0" />
            </nic>
         &win-saps;
         </server>
      </cluster>
      <cluster name="Auction WEBAPPL Servers" min-servers="1" max-servers="9"
               managed="true" is-device-model="Simulator" locale="en_US">
         <with-load-balancer name="Load Balancer 1"/>
         <server name="OAwebApplSrv01" locale="en_US">
            <nic managed="false" connected-to-switch="CISCO1"
                 connected-to-module="fa0" connected-to-port="22">
               <network-interface name="Management" ipaddress="9.3.5.157"
                        netmask="255.255.255.0" management="true"/>
            </nic>
         &win-saps;
         </server>
```

```
      </cluster>
      <cluster name="Auction DB Servers" min-servers="1" max-servers="5"
               managed="false" is-device-model="Simulator"  locale="en_US">
         <with-load-balancer name="Load Balancer 1"/>
         <server name="OADBSrv01"  locale="en_US">
            <nic connected-to-switch="CISCO1" connected-to-module="fa0"
                 connected-to-port="21" management="true">
               <network-interface name="Local Area Connection"
                        ipaddress="9.3.5.156" netmask="255.255.255.0" />
            </nic>
         &win-saps;
         </server>
      </cluster>
      <objective-analyzer  type-name="TIO capacity-on-demand"/>
   </application>
</customer>
```

Later in this paper we define a server template on page 52 and software definitions on page 43. We will then be associating these definitions with the Auction WEB Servers application tier so that the correct software and configuration settings are applied to the servers that will eventually get provisioned to this tier of the application.

Resource pool definitions

Resource pools are a core part of any provisioning operation. A resource pool is a group of un-allocated servers assigned to a specific application tier or tiers. When it is deemed necessary, either manually or by the workload manager, applications can receive the resources they need by allocating one or more of the available servers to the application tier in need. This will cause IBM Tivoli Provisioning Manager to update the status of those resources to allocated in the DCM, preventing them from being used elsewhere. When the workload monitor or another process decides they are no longer needed, they can be de-provisioned and returned to the resource pool to be used again.

For our scenario, the TOMCAT Pool resource pool will be defined as a pool of physical servers which are available for deployment into our Auction WEB Servers application tier. All servers in the TOMCAT Pool will have HTTP Server application installed for quick provisioning of the Auction application.

Below is the XML we used to define the TOMCAT Pool to our Data Center Model:

Example 3 Resource pool definition.

```
<spare-pool name="TOMCAT Pool" os-type="windows" locale="en_US">
   <server name="prov005" locale="en_US">
```

```
      <nic managed="false" connected-to-switch="CISCO1"
               connected-to-module="fa0" connected-to-port="6">
         <network-interface name="Management" ipaddress="9.3.5.25"
                  netmask="255.255.255.0" management="true"/>
      </nic>
      &win-saps;
   </server>
   <server name="prov006" locale="en_US">
      <nic managed="false" connected-to-switch="CISCO1"
               connected-to-module="fa0" connected-to-port="7">
         <network-interface name="Management" ipaddress="9.3.5.222"
                  netmask="255.255.255.0" management="true"/>
      </nic>
      &win-saps;
   </server>
   <server name="prov007" locale="en_US">
      <nic managed="false" connected-to-switch="CISCO1"
               connected-to-module="fa0" connected-to-port="8">
         <network-interface name="Management" ipaddress="9.3.5.31"
                  netmask="255.255.255.0" management="true"/>
      </nic>
      &win-saps;
   </server>
   <server name="prov010" locale="en_US">
      <nic managed="false" connected-to-switch="CISCO1"
               connected-to-module="fa0" connected-to-port="12">
         <network-interface name="Management" ipaddress="9.3.5.207"
                  netmask="255.255.255.0" management="true"/>
      </nic>
      &win-saps;
   </server>
</spare-pool>
```

Later, the servers in the TOMCAT Pool resource pool will be associated with a server template which has a software stack associated with it. This software stack contains the software module for the Apache Tomcat software. This will ensure that the servers have already been prepared, based on that template, with the required basic software for our Auction application. This allows for much faster deployment when the need arises, but also keeps the resource pool generic enough to allow the servers to be used for many different situations.

Software Definitions

Software Definitions are used in the DCM to identify basic information about a piece of software, for example: operating systems, software patches, and software products. This is known as a Software Module in the DCM XML file. You can then associate one or more installable files with the Software Definition.

These installables will contain information about where in the software repository the correct installation files are stored. In cases where multiple installable files are configured, it is necessary to define adequate requirements to ensure IBM Tivoli Provisioning Manager selects the correct installable file for deployment.

You can also include more granular information, called requirements, including dependencies and options such as Operating System family and version, to ensure deployment will occur on adequate hardware for the particular module.

Software Definitions also allow the definition of software configuration templates, also know as software resource templates (SRT) in the DCM XML definition, which provide information needed to install and configure software on the target server.

Software configuration templates can contain many parameters for both the installation and for the runtime of the product as well. and allow for a hierarchical definition. The parent template *must* be of type Installation. Child templates may be defined under a parent template to add additional features such as the ability to start and stop instances of installed software and can be of many types, for example Installation, Instance, Configuration, or Application data.

The name of the Software Definitions used in our scenario for the Auction application are shown below:

- Auction APPL
- Apache Tomcat Server

Auction APPL

The Auction application is the core of our scenario. The Auction APPL software definition identifies the product that will be deployed to all servers part of the Auction WEB Servers application tier. Servers that are dedicated to the Auction WEB Servers application tier will have the Auction application pre-installed. Additional servers provisioned to the application tier from its respective resource pool will have the Auction application deployed to them during the provisioning operation. Also, in case a new version of the Auction application is made available in the IBM Rational ClearCase software repository, it will be deployed to all servers that are part of the application tier at any given time.

All of the information required to successfully deploy the Auction application is defined in the software configuration template section. Specifically, we defined a parent software configuration template of type installation and placed all parameters for the existing Auction WEB Servers application tier.

For example, we define database configuration information such as database server hostname, port number, and instance owner. We also provide JDBC™

configuration attributes such as the location of the JDBC driver, and the name of the data source.

Another important piece of configuration information defined in the software configuration template is the package name, installation path, and staging area location. These will be used during workflows execution time for the installation of Auction application.

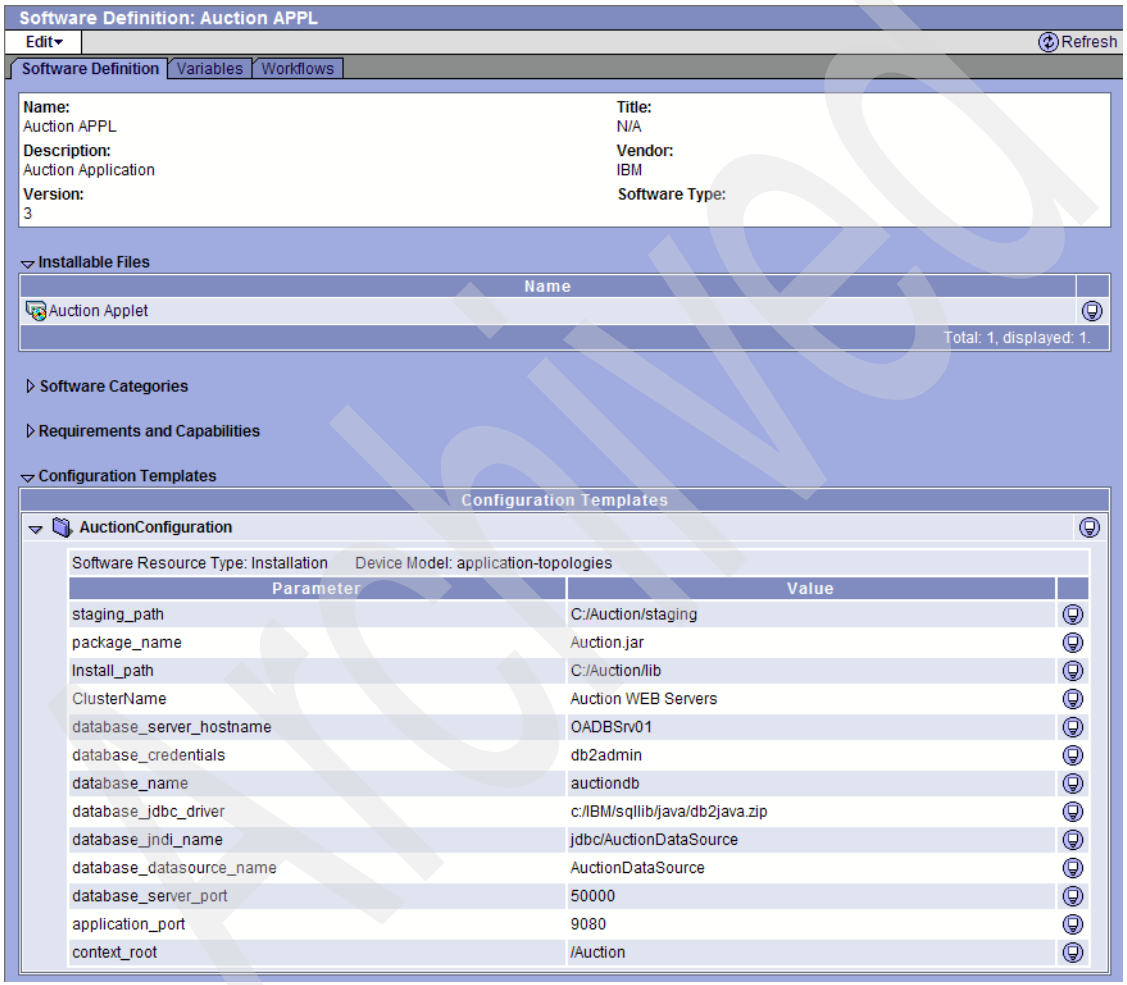

Figure 34 Auction software definition and configuration template

Under the Installable Files area, you see the software installable associated to the software definition. This software installable identify the installation context for the Auction application. As shown in the figure below, the Auction Applet software installable definition provides the name of the software repository in

which the installables for the application reside. This is shown in Figure 35 on the File Repository tag. In our scenario this definition points to the IBM Rational ClearCase software repository.

Figure 35 Software installable definition

Later in this paper, we create an automation package that provides operations for manipulating this software definition. Once the automation package is installed on the IBM Tivoli Provisioning Manager server, it defines a new device driver that must be associated to the Auction Applet software installable part of the software definition. The device driver will implement a logical operation and workflow implementation for installing Auction application on target servers.

The following example shows the entire software definition structure for the Auction application in XML format. This includes software definition, software installable, and software configuration template definitions.

Example 4 Auction software definition in XML format

```
<software-module name="Auction APPL" version="3" vendor="IBM" description="Auction Application"
is-draft="false">

   <installable-package name="Auction Applet" locale="en_US" description="Installable
definition for Auction APPL" version="1.2.0" priority="1" file-repository="ClearCaseRepository"
status="tested">
      <file name="Auction.jar" path="/Auction" />
   </installable-package>

   <software-resource-template name="AuctionConfiguration"
            software-resource-type="INSTALLATION"
            software-resource-device-model="application-topologies" multiplicity-type="One"
            software-configuration-type="Regular" is-selected="true">

      <template-param name="ClusterName" value="Auction WEB Servers" is-changeable="true"/>
      <template-param name="database_server_hostname" value="OADBSrv01" is-changeable="true"/>
      <template-param name="database_credentials" value="db2admin" is-changeable="true"/>
      <template-param name="database_name" value="auctiondb" is-changeable="true"/>
      <template-param name="database_jdbc_driver " value="c:/IBM/sqllib/java/db2java.zip"
                        is-changeable="true"/>
```

```
      <template-param name="database_jndi_name" value="jdbc/AuctionDataSource"
                      is-changeable="true"/>
      <template-param name="database_datasource_name" value="AuctionDataSource"
                      is-changeable="true"/>
      <template-param name="database_server_port" value="50000" is-changeable="true"/>
      <template-param name="application_port" value="9080" is-changeable="true"/>
      <template-param name="context_root" value="/Auction" is-changeable="true"/>
      <template-param name="Install_path" value="C:/Auction/lib" is-changeable="true"/>
      <template-param name="package_name" value="Auction.jar" is-changeable="true"/>
      <template-param name="staging_path" value="C:/Auction/staging" is-changeable="true"/>
   </software-resource-template>

</software-module>
```

Apache Tomcat Server

Apache Tomcat is required for our scenario, as it is the engine for running our Auction application. This is the definition for the Apache Tomcat software which are deployed to all dedicated servers of the Auction WEB Servers application tier as well servers in the TOMCAT Pool resource pool.

The definition used in this scenario is based on the template provided by the Apache automation package. The configuration template for this product contains information to enable workflows to collect information about the installation on those servers.

The following figure shows our case study scenario software definition for Apache Tomcat and its required installation configuration template.

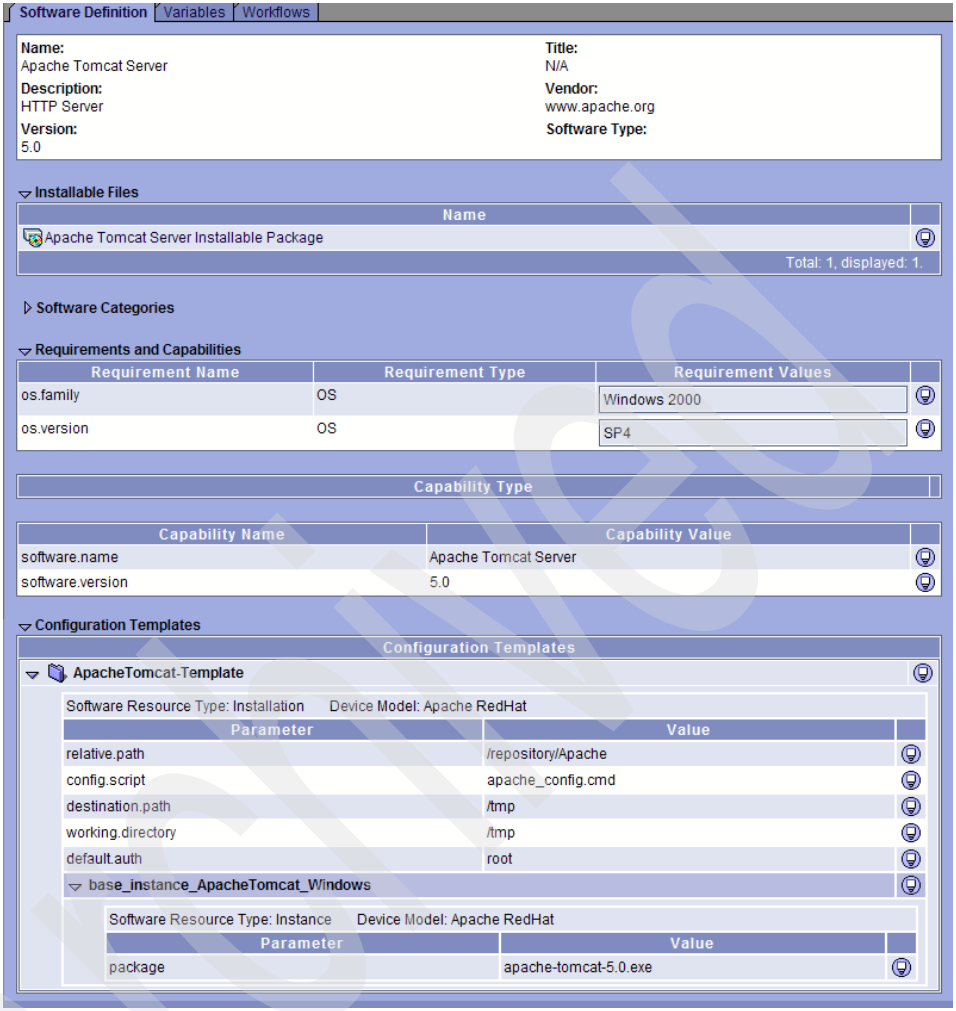

Figure 36 Apache Tomcat software definition

The following example shows the Apache Tomcat software definition in XML format.

Example 5 Apache Tomcat software definition in XML format

```
<software-module name="Apache Tomcat Server" locale="en_US" is-device-model="Apache RedHat"
version="5.0" vendor="www.apache.org" description="HTTP Server" is-draft="false">
    <software-capability name="software.name" value="Apache Tomcat Server" />
    <software-capability name="software.version" value="5.0" />

    <software-requirement name="os.family" type="OS" enforcement="MANDATORY" hosting="false"
                          accept-non-existing="true">
```

```
          <software-requirement-value value="Windows 2000" />
    </software-requirement>
    <software-requirement name="os.version" type="OS" enforcement="MANDATORY" hosting="false"
                          accept-non-existing="true">
          <software-requirement-value value="SP4" />
    </software-requirement>

    <installable-package name="Apache Tomcat Server Installable Package "
is-device-model="Apache RedHat" locale="en_US" description="Apache Tomcat HTTP Server"
version="5.0" priority="1" status="tested">
      <file name="apache-tomcat-5.0.exe" locale="en_US" path="/repository/Apache" />
      <property component="KANAHA" name="full.name" value="apache-tomcat-5.0.exe" />
      <property component="KANAHA" name="repository.name" value="Hook" />
      <software-requirement name="os.family" type="OS" enforcement="MANDATORY" hosting="false"
accept-non-existing="true">
        <software-requirement-value value="Windows 2000" />
      </software-requirement>
    </installable-package>

    <software-resource-template name="ApacheTomcat-Template"
software-resource-type="INSTALLATION" software-resource-device-model="Apache RedHat"
multiplicity-type="N" software-configuration-type="Regular" is-selected="true">
      <software-resource-template name="base_instance_ApacheTomcat_Windows"
software-resource-type="INSTANCE" software-resource-device-model="Apache RedHat"
multiplicity-type="N" software-configuration-type="Regular" is-selected="true">
        <template-param name="package" value="apache-tomcat-5.0.exe" is-changeable="true" />
      </software-resource-template>
      <template-param name="relative.path" value="/repository/Apache" is-changeable="true" />
      <template-param name="config.script" value="apache_config.cmd" is-changeable="true" />
      <template-param name="destination.path" value="/tmp" is-changeable="true" />
      <template-param name="working.directory" value="/tmp" is-changeable="true" />
      <template-param name="default.auth" value="root" is-changeable="true" />
    </software-resource-template>

</software-module>
```

Software Stacks

A software stack is a grouped set of software, organized in the correct installation order. IBM Tivoli Provisioning Manager uses software stacks to install multiple products, patches, or even other stacks onto servers to prepare them for provisioning into production.

Each module when included in a software stack will have a cloned version of the original software configuration template defined in the software stack entry in the DCM. This allows for additional customization of the software in relation only to the particular stack you are currently concerned with. As the cloned configuration

template and the original configuration template have unique IDs in the DCM, changes made to one will not be reflected in the other. In other words, changing attributes in the configuration template of the software stack will not change the configuration template defined in the software module definition.

For our scenario we defined two software stacks to support our sample application environment:

- A software stack to be associated to the server template of the TOMCAT Pool resource pool. This ensures that servers in the resource pool are in a ready state for deployment into the Auction application environment faster by having most of the requisite software installed for our particular scenario.
- A second software stack, named AUCTION_Software_Stack, to be associated to the server template of our Auction WEB Servers application tier. This software stack has all the software required for the installation and configuration of the Auction application.

In our case study scenario environment, the following software definitions are assigned to the AUCTION_Software_Stack:

- Apache Tomcat Server
- Auction APPL

The following figure shows the software stack definition and associated software required for Auction application, along with the iterator installable. The iterator is a function in IBM Tivoli Provisioning Manager that will allow multiple pieces of software to be installed by “looping” through the list of modules in the stack. This is the reason the objects need to be added to the stack in the correct installation order. The iterator is only required when multiple modules have been associated with a software stack.

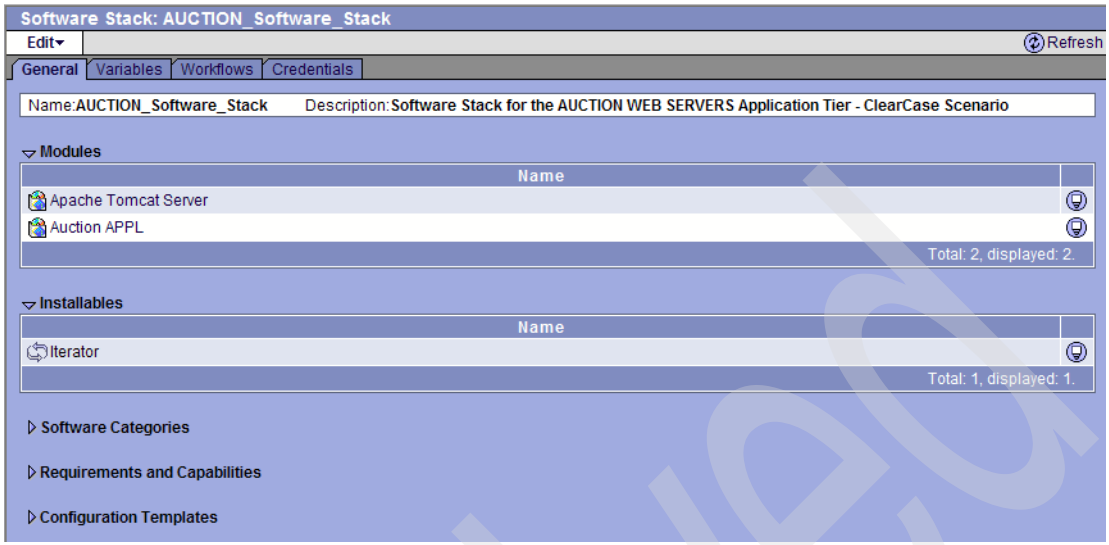

Figure 37 The AUCTION_Software_Stack

Once we have created the software stack for Auction application, and associated the required software definitions to it, we will see the configuration template for the software stack that IBM Tivoli Provisioning Manager creates by cloning the parent templates originally created for the software definition. At this time, if required, these templates could be customized to reflect the needs of this particular stack.

However, if a required parameter is altered or added to the configuration template for a software definition after it has been added to a software stack, changes made in the original configuration template will not be present on the cloned template for the software associated to the stack. It is important to remember this as you move forward creating software stacks for your particular environment, as changes to these parent templates will require you to remove and re-add the effected software definition to your previously created software stacks if you want the changes to exist in them as well.

Below, in Figure 38 on page 52, part of the configuration template for our AUCTION_Software_Stack is shown. All of the listed parameters were acquired in the cloning of the template from the parent software definition we created. Notice the name of the configuration templates listed all have numbers appended to the names, indicating they are clones of the original. This is where the configurations for each software stack can be edited without impact to any parent or other software stack using a clone of the same parent.

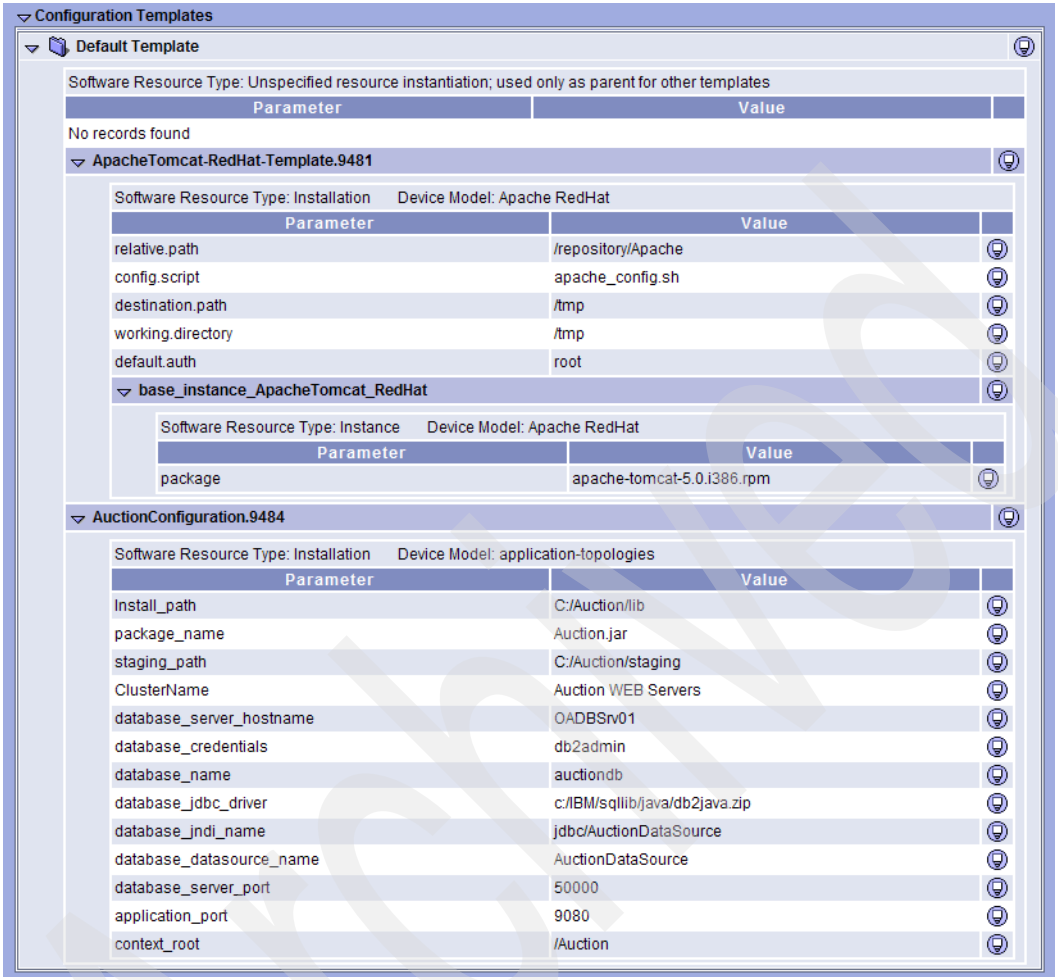

Figure 38 Cloned configuration template for AUCTION_Software_Stack

Define Server Templates

Server templates are used in IBM Tivoli Provisioning Manager to provide software, storage, and network information to the manager so that once servers are associated with a template for a resource pool or application tier, the software applications and software configurations provided by the template will be pre-installed and configured as defined by the template, putting the server in the desired state for provisioning based on the location of the server.

For our Auction application scenario, we defined two server templates:

- A server template for the application tier Auction WEB Servers of the WEB Auction application. This server template will have the AUCTION_Software_Stack software stack defined in the previous section associated to it.
- A second server template for the TOMCAT Pool resource pool.

The server template shown in the following figure shows the AUCTION_Software_Stack defined in the Data Center Model. Under Software Definitions, you will see how we have associated the AUCTION_Software_Stack to the server template, confirming that any server brought into the application tier will have the modules and configurations defined by that software stack applied to them. Notice also that if required, routes and network information could be provided here as well. Storage templates can also be associated here if required.

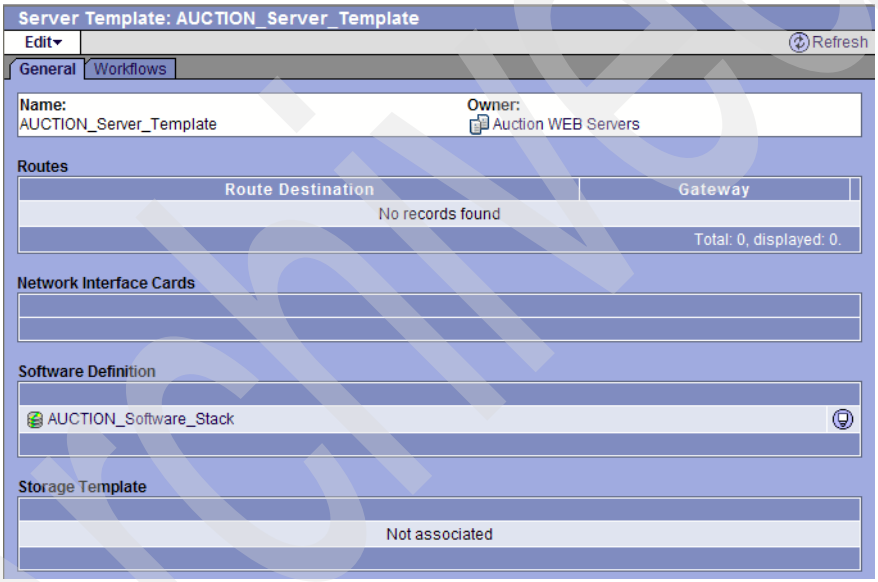

Figure 39 AUCTION_Server_Template

Also on the figure above, under the Workflows tab, we will associate the device driver for the provisioning operations. This device driver will be part of an automation package created exclusively for our case study scenario. The following sections of this paper concentrate on the development of this automation package.

Define Software Repository

IBM Tivoli Provisioning Manager allows the definition of external specialized software repositories.

In IBM Tivoli Provisioning Manager terms, an external software repository is a server that maintains a current library of software, software patches, and their associated installation information.

Software repositories are defined in IBM Tivoli Provisioning Manager using a file repository DCM definition. As any server defined in the DCM, file repositories must have an IP address, a network interface card and network interfaces definition, service access points defined, workflows assigned to the server by selecting the device driver for the file repository. On top of those attributes, a file server definition must contain the path to the software installables.

As described in “Setting up IBM Rational ClearCase software repository” on page 11, the development environment of our sample Auction application makes use of a staging area in which new versions of the application are stored. We use this staging area defined in the IBM Rational ClearCase development environment as the path for our file repository.

The following example shows the file repository definition in XML format.

Example 6 File repository - XML definition

```
<file-repository name="ClearCaseRepository" locale="en_US"
root-path="F:\ClearCase_Stage" ipaddress="9.3.5.25" >

   <network-interface name="Management" ipaddress="9.3.5.25"
                      netmask="255.255.255.0" management="false"
                      failed="false" />

    <sap name="SSH-Server" port="22" host="true" is-device-model="SSH Service
              Access Point" app-protocol="SSH" locale="en_US">
      <credentials search-key="primary" is-default="true">
         <rsa-credentials username="Administrator" />
      </credentials>
      <default-sap operation-type="execute-command" />
   </sap>
   <sap name="SSH-Client" port="0" host="false" is-device-model="SSH Service
              Access Point" app-protocol="SSH" locale="en_US">
      <credentials search-key="primary" is-default="true">
         <rsa-credentials username="Administrator" />
      </credentials>
   </sap>
   <sap name="SCP-Server" port="22" host="true" is-device-model="SSH Service
              Access Point" app-protocol="SCP" locale="en_US">
      <credentials search-key="primary" is-default="true">
         <rsa-credentials username="Administrator" />
      </credentials>
      <default-sap operation-type="file-transfer" />
   </sap>
```

```
   <sap name="SCP-Client" port="0" host="false" is-device-model="SSH Service
             Access Point" app-protocol="SCP" locale="en_US">
      <credentials search-key="primary" is-default="true">
         <rsa-credentials username="Administrator" />
      </credentials>
   </sap>

</file-repository>
```

The following figure shows the file repository after importing the above XML definition into the IBM Tivoli Provisioning Manager DCM.

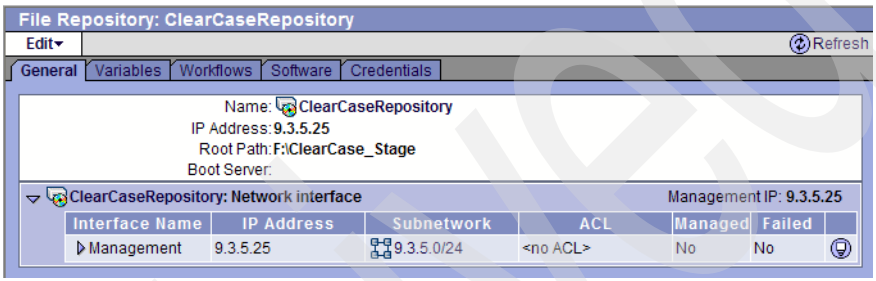

Figure 40 ClearCase file repository

Workflow design and development

Workflows are a key component for provisioning. They can be used for gathering information about an instance, installation, a piece of hardware, or any other object defined to the DCM which will be acted upon. A workflow can then use this information to perform defined tasks. Workflows are simple structured programs with a number of constructs to manage data center components. They can be very complex and long, in which a workflow executes many other workflows to perform the desired operation, or they can be as simple as one command to check the status of a machine with a ping. It is critical that workflows be planned, defined and structured correctly to ensure provisioning can occur in support of a predefined working process.

Note: It is not our intention to teach the reader how to develop workflows in this paper. This is beyond the scope of this paper. For more information about developing workflows, refer to the redbook *Developing Workflows and Automation Packages for IBM Tivoli Intelligent Orchestrator,* SG24-6057; and the *IBM Tivoli Provisioning Manager V3.1 Workflow Developer's Guide,* GC32-1662.

For our scenario, there are a number of workflows required to install our sample application using the IBM Rational ClearCase software repository. Some workflows will gather required information from the Data Center Model such as server IDs, resource template IDs, and configuration parameters from software definitions, software repository information, and so forth.

The list below shows the workflows used in our scenario and describes the function performed by each one. These workflows will be part of an automation package for Auction application we will be creating and installing later in the paper.

_AUCTION_GetInstallableInformation

This workflow collects software attributes from the configuration template defined for the Auction application software definition (Auction APPL) as well as the ID of the ClearCaseRepository file repository defined for the Auction application software installable definition (Auction Applet).

Example 7 _AUCTION_GetInstallableInformation workflow

```
workflow _AUCTION_GetInstallableInformation(in SoftwareID, out
fileRepositoryId, out install_path, out package_name, out staging_path, out
context_root) LocaleInsensitive

   var AUCTIONtemplateid

   # Check the objectType to assure this is a Software Product
   var objectType = DCMQuery(/dcmObject[@id=$SoftwareID]/dcmobjecttype/@name)
   if Jython[ objectType != "SOFTWARE_PRODUCT" ] then
     throw ErrorAuctionInstall "ERROR: The given SoftwareID is not an
installable"
   endif
   log info Jython("objectType: " + objectType)

   # Collects File repository ID of software installable
   fileRepositoryId =
DCMQuery(/softwareinstallable[@id=$SoftwareID]/filerepository)
   log info fileRepositoryId

   # Collect Template IDs of Auction software stack
   array parameters =
DCMQuery(/softwaremodule[@name="AUCTION_Software_Stack"\]/softwareresourcetempl
ate/childsoftwareresourcetemplate/@id)
   log info parameters

   # Collect ID of Auction software resource template
   foreach paramid in parameters do
      var parametername =
DCMQuery(/childsoftwareresourcetemplate[@id=$paramid\]/@name)
```

```
        log info parametername

        if Jython(parametername == "AuctionConfiguration." + paramid) then
          AUCTIONtemplateid = paramid
        endif
    done

    log info AUCTIONtemplateid

    # Collect Attributes of Auction software resource template
    array parameterids =
DCMQuery(/templateparam[@templateid=$AUCTIONtemplateid\])
    log info parameterids

    foreach paramid in parameterids do
     var parametername = DCMQuery(/templateparam[@id=$paramid\]/@name)
        var parametervalue =
Java[com.ibm.tivoli.orchestrator.datacentermodel.helper.SoftwareHelper#getTempl
ateParamValueById(paramid)]
        log info parametername
        if Jython(parametername == "Install_path") then
          install_path = parametervalue
        endif
        if Jython(parametername == "package_name") then
          package_name = parametervalue
        endif
        if Jython(parametername == "staging_path") then
          staging_path = parametervalue
        endif
        if Jython(parametername == "context_root") then
          context_root = parametervalue
        endif
    done
    log info install_path
    log info package_name
    log info staging_path
    log info context_root
```

_AUCTION_GetFileRepositoryInformation

This workflow receives the file repository ID from the _AUCTION_GetInstallableInformation workflow and collects attributes from the file repository defined for the Auction application software installable definition (Auction Applet). This workflow returns the location of the Auction application software repository.

Example 8 _AUCTION_GetFileRepositoryInformation workflow

```
workflow _AUCTION_GetFileRepositoryInformation(in fileRepositoryId, in
context_root, out source_path)  LocaleInsensitive

   # Collect the Root path for the software repository
   var repositoryRoot =
DCMQuery(/filerepository[@id=$fileRepositoryId]/@rootpath)

   log info repositoryRoot

   # Source path variable = root path + context root
   source_path = Jython(repositoryRoot + context_root)
   log info source_path

   # Convert file path to cygwin path format
   source_path =
Java[com.thinkdynamics.kanaha.util.PathHelper#convertToCygwinPath(source_path)]
   log info source_path
```

_AUCTION_GetFileFromRepository

This workflow collects the Auction application installable files from the IBM Rational ClearCase software repository and transfers them to a staging area on the target server. This is the preparation step for the actual installation process in our case study scenario. This workflow was developed based on the FileRepository_GetFile_SCP workflow which is provided by IBM Tivoli Provisioning Manager V3.1.

Example 9 _AUCTION_GetFileFromRepository workflow

```
workflow _AUCTION_GetFileFromRepository(in fileRepositoryId, in DeviceID, in
packageName, in sourcePath, in stagingPath) LocaleInsensitive

  var TimeoutInSeconds = "300"

  # A source path of / is the root directory, prepend nothing
  if Jython(sourcePath == "/") then
    sourcePath = ""
  endif

  # A destination path of / is the root directory, prepend nothing
  if Jython(stagingPath == "/") then
    stagingPath = ""
  endif

  # Get the source and the destination name
  var fileRepositoryName =
DCMQuery(/managedsystem[@id=$fileRepositoryId\]/@name)
```

```
  var destServerName = DCMQuery(/managedsystem[@id=$DeviceID\]/@name)

  # Log what we are going to do:
  log info Jython("Copy File:  " + sourcePath + "/" + packageName + "@" +
fileRepositoryName + "  to  " + stagingPath + "/" + packageName + "@" +
destServerName )

  # Reset path to original
  if Jython(sourcePath == "") then
    sourcePath = "/"
  endif

  if Jython(stagingPath == "") then
    stagingPath = "/"
  endif

  # Find the SAP id of the servers
  var CopyHostSAPID
  var ExecuteHostSAPID
  Get_Default_SAP_For_Operation(fileRepositoryId, "file-transfer",
CopyHostSAPID)
  Get_Default_SAP_For_Operation(DeviceID, "execute-command", ExecuteHostSAPID)

  # Convert any Windows file path to cygwin path format
  sourcePath =
Java[com.thinkdynamics.kanaha.util.PathHelper#convertToCygwinPath(sourcePath)]
  stagingPath =
Java[com.thinkdynamics.kanaha.util.PathHelper#convertToCygwinPath(stagingPath)]

  # Test the ServiceAccessPoint.CopyFile() input parameters
  if Jython( CopyHostSAPID == None or ExecuteHostSAPID == None or sourcePath ==
None or packageName == None or stagingPath == None or CopyHostSAPID == "" or
ExecuteHostSAPID == "" or sourcePath == "" or packageName == "" or stagingPath
== "") then
    throw MissingInputParameter "Input parameters cannot be null"
  endif

  # Set ClientIsSource to FALSE for the Get <file> operation
  var Client_Is_Source = "false"
  var Final_Copy_Credentials_key
  var Final_Execution_Credentials_key

  java:com.thinkdynamics.kanaha.de.javaplugin.sap.MatchFileTransferSaps(Client_
Is_Source, "default", CopyHostSAPID, DeviceID, "default", ExecuteHostSAPID,
Final_Copy_Credentials_key, Final_Execution_Credentials_key, fileRepositoryId)
```

```
  ServiceAccessPoint.CopyFile(CopyHostSAPID, Final_Copy_Credentials_key,
ExecuteHostSAPID, Final_Execution_Credentials_key, sourcePath, packageName,
stagingPath, packageName, Client_Is_Source, TimeoutInSeconds)
```

_AUCTION_PerformInstall

This workflow performs the installation process of our case study scenario. It receives the staging area location, package name and install path of our sample Auction application from the _AUCTION_GetInstallableInformation workflow and executes the installation command. In our case, the installation process resumes to backing up the current Auction application is as simple as copying the installable files from the staging are to the installation path of Auction application.

Example 10 _AUCTION_PerformInstall workflow

```
workflow _AUCTION_PerformInstall (in DeviceID, in staging_path, in
package_name, in install_path, out ReturnCode, out ReturnErrorString, out
ReturnResult) LocaleInsensitive

   var TimeoutInSeconds = "300"
   var TimeOutAs = "error"
   var credentials = "primary"

   # Convert file paths to cygwin paths format
   staging_path =
Java[com.thinkdynamics.kanaha.util.PathHelper#convertToCygwinPath(staging_path)
]
   install_path =
Java[com.thinkdynamics.kanaha.util.PathHelper#convertToCygwinPath(install_path)
]

   # Builds the backup command
   var backup_command = Jython("cp " + install_path + "/" + package_name + " "
+ install_path + "/" + package_name + ".BACKUP")
   log info backup_command

   # Executes the backup command
   Device.ExecuteCommand (DeviceID, backup_command, "/", credentials,
TimeoutInSeconds, TimeOutAs, ReturnCode, ReturnErrorString, ReturnResult )

   # Tests backup success
   if Jython(ReturnCode != "0") then
    log info "Backup of Auction FAILED !"
      throw AnException
   endif

   # Builds the install command
   var install_command = Jython("cp " + staging_path + "/" + package_name + "
" + install_path  + "/" +  package_name)
```

```
   log info install_command

   # Executes the install command
   Device.ExecuteCommand (DeviceID, install_command, "/", credentials,
TimeoutInSeconds, TimeOutAs, ReturnCode, ReturnErrorString, ReturnResult )
```

_AUCTION_CompleteProcess

This workflow provides the framework for the entire provisioning process of the Auction application into the existing application cluster.

This workflow executes the workflows described in the previous sections in the proper order, and implements the SoftwareInstallable.Install logical operation.

Example 11 _AUCTION_CompleteProcess workflow

```
workflow _AUCTION_CompleteProcess(in SoftwareID, in DeviceID, in
SoftwareResourceTemplateID) implements SoftwareInstallable.Install
LocaleInsensitive

   var fileRepositoryId
   var install_path
   var package_name
   var source_path
   var staging_path
   var context_root
   var ReturnCode
   var ReturnErrorString
   var ReturnResult

   # Collects Software Installable information
   _AUCTION_GetInstallableInformation(SoftwareID, fileRepositoryId,
install_path, package_name, staging_path, context_root)

   # Collects Auction repository source path
   _AUCTION_GetFileRepositoryInformation(fileRepositoryId, context_root,
source_path)

   # Get the installable from the repository and transfer it to staging area
on the target server
   _AUCTION_GetFileFromRepository(fileRepositoryId, DeviceID, package_name,
source_path, staging_path)

   # Perform the installation of the new Auction build
   _AUCTION_PerformInstall (DeviceID, staging_path, package_name,
install_path, ReturnCode, ReturnErrorString, ReturnResult)

   if Jython(ReturnCode != "0") then
    log info "Installation of Auction FAILED !"
```

```
    throw AnException
endif
```

Automation Package assembly and installation

The current implementation of an automation package contains the binaries and metadata information that is related to specific objects in the DCM. In this scenario these objects represent the entire infrastructure defined for the Auction application cluster in our DCM. In this section we present the required tasks to create an automation package for provisioning and de-provisioning of Auction in our environment.

As per the *IBM Tivoli Provisioning Manager V3.1 Workflow Developer's Guide,* GC32-1662, guidelines, the following activities must be completed before creating the automation package for our Auction application.

1. Define the processes for installing the software and any other operations that will be required for provisioning and de-provisioning the application.
2. Perform the software definition, installable file, and the file repository that stores the software package in the DCM
3. Create the workflows for the identified procedures.
4. Determine the device models
5. Create the automation package

At this stage in our scenario we have already performed Steps 1, 2, and 3 above, as described in earlier sections.

The automation package has to provide a new device model for the Auction application. We named our device model `ITSO_AUCTION_Application` and chose to define this new device model under the Software Products device model category. The ITSO_AUCTION_Application device model implements the SoftwareInstallable.Install logical operation.

Now we are ready to create the automation package. To accomplish this task, we use the Automation Package Development Environment (APDE), a plugin for Eclipse. The directions for installing the APDE are available at:

`http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.tivoli.tio.doc/tio_nav.xml`

Once the APDE is installed, you can follow the instructions for creating a new project. When the project has been created, you can add workflows to the project either by creating them within the APDE, or by importing them from files you

have saved somewhere else if you have created them previously using a text editor or the IBM Tivoli Provisioning Manager V3.1 Workflow Composer.

Once all of the required files for your automation package are present within the project and tested, you can build your automation package file. You begin this process by right-clicking on the build.xml file in your project, selecting run, and then selecting the *second* Ant Build from the menu.

Once the build XML file window pops up, as shown in Figure 41, you can select the required options and attributes, and then click run to build the automation package file. Figure 41 also shows the APDE with our workflows, and all the files that will be packaged into the automation package for the scenario presented in this paper.

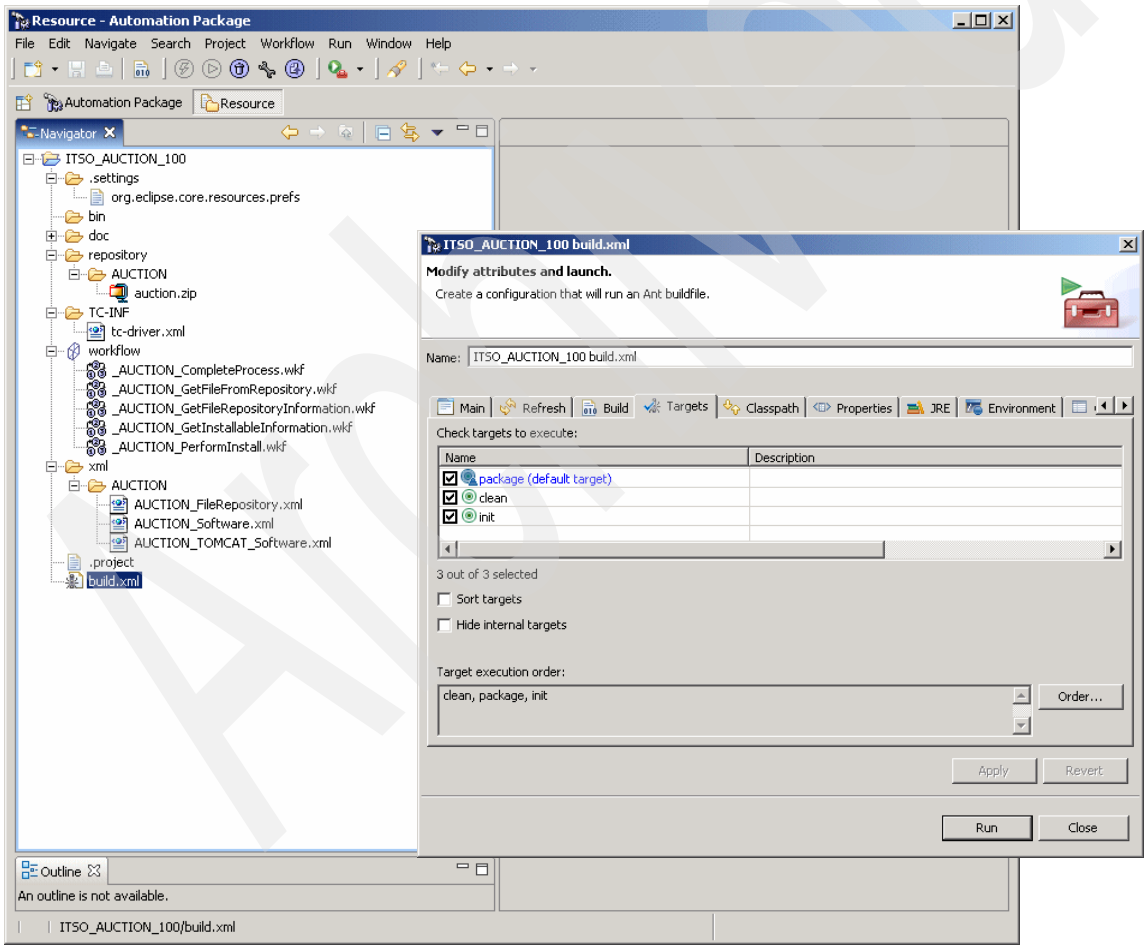

Figure 41 Automation Package Development Environment

In the next figure, Figure 42, we show the APDE workspace post Ant Build process. You can now see that in the left hand pane, at the bottom of our project, we have an automation package for Auction application, named ITSO_AUCTION_100.tcdriver.

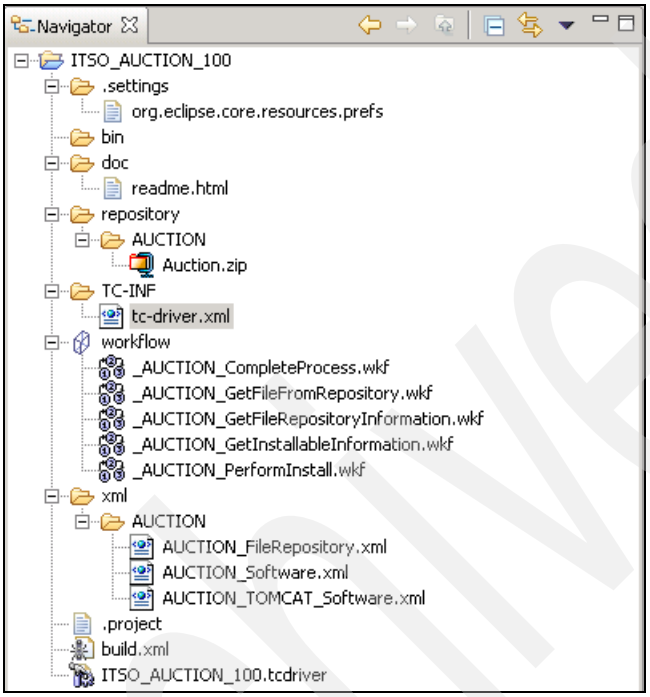

Figure 42 APDE workspace post build

The following example shows the manifest files (tc-driver.xml) for our automation package. This file will identify all the requirements and characteristics of the automation package. It lists the scripts, workflows, and any other files you need to include with the installation of the automation package and the final destination of the files. It also includes the device model definitions that need to be implemented by this automation package.

Example 12 Auction automation package manifest file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tc-driver PUBLIC "-//IBM//DTD TC Driver 2.0//EN"
"http://www.ibm.com/tivoli/orchestrator/dtd/tcdriver.dtd">

<tc-driver>
  <tc-driver-format>2.0</tc-driver-format>
  <driver-name>ITSO_AUCTION_100</driver-name>
  <version>1.0</version>
```

```
<description />
<documentation location="doc/readme.html" />
<dependencies>
  <dependency name="apache" />
</dependencies>
<property name="tc.pkg" location="com.thinkdynamics.kanaha.tcdrivermanager.action" />
<actions>
  <action name="copy-file" class="${tc.pkg}.CopyFileActions" />
  <action name="java-plugin" class="${tc.pkg}.JavaPluginActions" />
  <action name="workflow" class="${tc.pkg}.TxtWorkflowAction" />
  <action name="import" class="${tc.pkg}.ImportAction" />
</actions>

<items>
  <item name="repository/AUCTION/Auction.zip" action="copy-file">
    <param name="dest.path" value="${tc.home}/repository/AUCTION/Auction.zip" />
    <param name="chmod" value="644" />
  </item>
  <item name="xml/AUCTION/AUCTION_TOMCAT_Software.xml" action="copy-file">
    <param name="dest.path" value="${tc.home}/xml/AUCTION/AUCTION_TOMCAT_Software.xml" />
    <param name="chmod" value="644" />
  </item>
  <item name="xml/AUCTION/AUCTION_Software.xml" action="copy-file">
    <param name="dest.path" value="${tc.home}/xml/AUCTION/AUCTION_Software.xml" />
    <param name="chmod" value="644" />
  </item>
  <item name="xml/AUCTION/AUCTION_FileRepository.xml" action="copy-file">
    <param name="dest.path" value="${tc.home}/xml/AUCTION/AUCTION_FileRepository.xml" />
    <param name="chmod" value="644" />
  </item>
  <item name="workflow/_AUCTION_GetInstallableInformation.wkf" action="workflow">
    <param name="editable" value="false" />
  </item>
  <item name="workflow/_AUCTION_GetFileRepositoryInformation.wkf" action="workflow">
    <param name="editable" value="false" />
  </item>
  <item name="workflow/_AUCTION_GetFileFromRepository.wkf" action="workflow">
    <param name="editable" value="false" />
  </item>
  <item name="workflow/_AUCTION_PerformInstall.wkf" action="workflow">
    <param name="editable" value="false" />
  </item>
  <item name="workflow/_AUCTION_CompleteProcess.wkf" action="workflow">
    <param name="editable" value="false" />
  </item>
</items>

<device-models>
  <device-model name="ITSO_AUCTION_Application" category="Software Products">
```

```
      <workflow name="_AUCTION_CompleteProcess" />
    </device-model>
  </device-models>

  <dcm>
    <item name="xml/AUCTION/AUCTION_TOMCAT_Software.xml" action="import" />
    <item name="xml/AUCTION/AUCTION_Software.xml" action="import" />
    <item name="xml/AUCTION/AUCTION_FileRepository.xml" action="import" />
  </dcm>

</tc-driver>
```

The automation package file now can be installed onto a working IBM Tivoli Provisioning Manager V3.1 server for use. This automation package file can be installed either using the APDE or using the automation package manager command **%TIO_HOME%\tools\tc-driver-manager.cmd installDriver <packageName>**, as shown in the following example.

Example 13 Automation package installation

```
C:\IBM\tivoli\thinkcontrol\tools>tc-driver-manager.cmd installDriver ITSO_AUCTION_100
2005-10-17 16:22:14,609 INFO  COPTDM001I TCDrivermanager was started.
2005-10-17 16:22:14,984 INFO  COPTDM004I Config directory:
"file:C:\IBM\tivoli\thinkcontrol\config/".
2005-10-17 16:22:15,000 INFO  COPTDM004I Config directory:
"file:C:\IBM\tivoli\thinkcontrol\config/".
2005-10-17 16:22:15,047 INFO  COPTDM002I Driver directory: "Driver directory:
"C:\IBM\tivoli\thinkcontrol/drivers/".".
2005-10-17 16:22:15,047 INFO  COPTDM003I Installing driver "ITSO_AUCTION_100", force: "false",
installItems: "true".
2005-10-17 16:22:18,750 INFO  COPTDM006I Creating DCM_OBJECT entry for TCDriver "ITSO_AUCTION_100".
2005-10-17 16:22:18,875 INFO  COPTDM011I Installing driver item "repository/AUCTION/Auction.zip".
2005-10-17 16:22:18,906 INFO  creating directory 'C:\IBM\tivoli\thinkcontrol\repository\AUCTION'.
2005-10-17 16:22:18,906 INFO  creating file
'C:\IBM\tivoli\thinkcontrol/repository/AUCTION/Auction.zip'.
2005-10-17 16:22:18,922 INFO  copying data to file
'C:\IBM\tivoli\thinkcontrol/repository/AUCTION/Auction.zip'.
2005-10-17 16:22:19,984 INFO  COPTDM011I Installing driver item
"xml/AUCTION/AUCTION_TOMCAT_Software.xml".
2005-10-17 16:22:20,000 INFO  creating directory 'C:\IBM\tivoli\thinkcontrol\xml\AUCTION'.
2005-10-17 16:22:20,000 INFO  creating file
'C:\IBM\tivoli\thinkcontrol/xml/AUCTION/AUCTION_TOMCAT_Software.xml'.
2005-10-17 16:22:20,000 INFO  copying data to file
'C:\IBM\tivoli\thinkcontrol/xml/AUCTION/AUCTION_TOMCAT_Software.xml'.

2005-10-17 16:22:20,312 INFO  COPTDM011I Installing driver item "xml/AUCTION/AUCTION_Software.xml".
2005-10-17 16:22:20,344 INFO  creating file
'C:\IBM\tivoli\thinkcontrol/xml/AUCTION/AUCTION_Software.xml'.
2005-10-17 16:22:20,344 INFO  copying data to file
'C:\IBM\tivoli\thinkcontrol/xml/AUCTION/AUCTION_Software.xml'.
```

```
2005-10-17 16:22:20,547 INFO  COPTDM011I Installing driver item
"xml/AUCTION/AUCTION_FileRepository.xml".
2005-10-17 16:22:20,562 INFO  creating file
'C:\IBM\tivoli\thinkcontrol/xml/AUCTION/AUCTION_FileRepository.xml'.
2005-10-17 16:22:20,562 INFO  copying data to file
'C:\IBM\tivoli\thinkcontrol/xml/AUCTION/AUCTION_FileRepository.xml'.
2005-10-17 16:22:20,875 INFO  COPTDM011I Installing driver item
"workflow/_AUCTION_GetInstallableInformation.wkf".
2005-10-17 16:22:21,344 INFO  COPTDM011I Installing driver item
"workflow/_AUCTION_GetFileRepositoryInformation.wkf".
2005-10-17 16:22:22,406 INFO  COPTDM011I Installing driver item
"workflow/_AUCTION_GetFileFromRepository.wkf".
2005-10-17 16:22:22,625 INFO  COPTDM011I Installing driver item
"workflow/_AUCTION_PerformInstall.wkf".
2005-10-17 16:22:22,656 INFO  COPTDM011I Installing driver item
"workflow/_AUCTION_CompleteProcess.wkf".
2005-10-17 16:22:22,719 INFO  COPTDM012I Creating Device Model "ITSO_AUCTION_Application".
2005-10-17 16:22:22,750 INFO  COPTDM016I Creating the dcm object properties template for Device
Model "ITSO_AUCTION_Appl
ication".
2005-10-17 16:22:22,750 INFO  COPTDM017I Associating workflow "_AUCTION_CompleteProcess" with Device
Model "ITSO_AUCTION
_Application".
2005-10-17 16:22:22,766 INFO  COPTDM011I Installing driver item
"xml/AUCTION/AUCTION_TOMCAT_Software.xml".
2005-10-17 16:22:23,328 INFO  COPTDM011I Installing driver item "xml/AUCTION/AUCTION_Software.xml".
2005-10-17 16:22:23,438 INFO  COPTDM011I Installing driver item
"xml/AUCTION/AUCTION_FileRepository.xml".
2005-10-17 16:22:24,391 INFO  COPTDM005I TCDrivermanager was stopped.

Installation successful. (Driver name:ITSO_AUCTION_100)
```

After the automation package installation completes successfully, use the getDriverStatus option above to check the status again, verifying your install completed correctly.

There are also other files included with the automation package such as configuration templates, workflows, repository files, and documentation. It should be good practice to verify the existence of these as well after the automation package has been installed and passed verification as additional validation step.

Scenario execution

In this section we demonstrate the ability to use IBM Tivoli Provisioning Manager to deploy an application that has been developed and staged in a IBM Rational ClearCase environment. As described in “Scenario overview” on page 2, a problem with the Auction application is reported to the development team. The

development of a fixed version of the Auction application is performed using IBM Rational ClearCase, and a the new version is placed in the IBM Rational ClearCase software repository. At this point, the installation of the Auction application has been automated. IBM Tivoli Provisioning Manager has been configured to gather the newly created Auction application installables from the IBM Rational ClearCase repository and perform the deployment to the production environment.

In this section we provide information and screen captures which document the actual development and provisioning process of the fixed version of our sample Auction application.

The Auction application is running on the production environment is experiencing several crashes and interruptions. The interruptions have become so common that is affecting our customers business. Upon further investigation by the support team, it was found that the Auction application's main applet file was causing problems. The support team then proceeds to open a defect against the application in production. This starts the process in development to address the problem.

The following are activities that must be addressed in IBM Rational ClearCase development environment:

- “Change request qualification” on page 68
- “Development activities” on page 69
- “Integrate and build” on page 69

The deployment of the fixed version of Auction application is performed by the support team using IBM Tivoli Provisioning Manager and is presented in “Provisioning to the production environment” on page 69.

Change request qualification

Once the support team submits the defect against the application, the development lead of the Auction application starts to assess the impact of the change and validates whether the change is significant or small. Once the assessment is completed the development lead assigns the problem to a developer in their group with appropriate skills. The developer is notified of the assignment and the development process is initiated. In this scenario the Auction repository is already setup in IBM Rational ClearCase using the steps identified in “Setting up IBM Rational ClearCase software repository” on page 11.

The baseline from which the developer would be working has been recommended. The Change Request process could be automated using IBM Rational ClearQuest®. IBM Rational ClearQuest allows state based change request model to integrate with the development activities.

Development activities

The developer who has been notified about the problem assigned, starts to address the problem ticket. The developer proceeds to join the UCM project that addresses the Auction application in question and modifies artifacts based on the activity context which maps to the defect assigned. This was presented in “Make Changes” on page 28.

The developer uses the typical check-out, revise and check-in model to modify changes. Once the changes are done the developer rebases their development stream in the IBM Rational ClearCase environment before they start the unit testing process. This is presented in “Rebasing the development stream” on page 31.

The developer resolves some of the integration issues that might arise out of the rebase and run the unit tests against the changes that have been made. After the tests are complete it is now up to the developer to deliver their changes to integration stream of the Auction Project, so that the fix can be included in the next baseline that will be built and staged for the support teams to deploy to the target environment. This was presented in “Delivering the changes” on page 29.

Integrate and build

The development lead gets notified of the fix from the developer and starts to plan for the build of the fix. Using the IBM Rational ClearCase the development lead locks down the integration stream and performs a build using ANT. The ant script builds the binaries and checks the binaries in to ClearCase, creates a baseline and copies the file to a location where the support team can access it for deployment. This was presented in “Build and Stage development artifacts” on page 32.

As part of the build that was generated by ANT, one of the tasks was to copy the file to a source /staging area where IBM Tivoli Provisioning Manager could point to for transporting the source file. In our scenario the staging area is located on the F:\ClearCase_Stage directory. This staging area will serve as the software repository defined for our sample Auction application.

The resulting binaries contains information about the development environment to support trace-ability from deployment in case such a situation were to arise again.

Provisioning to the production environment

The support team is notified about the fix and depending on the target environment would deploy the fix for the target environment. Typically in most

cases the support team would first deploy to a System testing environment, then to a pre-production environment and finally to production. Once all the stages of testing are satisfied the problem ticket/change request would be closed.

In our scenario we show an execution of a provisioning operation of the fixed version of the Auction application to a server in the production environment.

As mentioned in “Software Definitions” on page 43, we also changed the Auction APPL software e definition by associating the device driver defined by the automation package developed in “Automation Package assembly and installation” on page 62. In the following figure, the device driver `ITSO_AUCTION_Application` is assigned to the to the Auction Applet software installable definitions of Auction APPL software module. This association enables IBM Tivoli Provisioning Manager to perform installations of Auction application using the SoftwareInstallable.Install logical operation, implemented by the _AUCTION_CompleteProcess workflow.

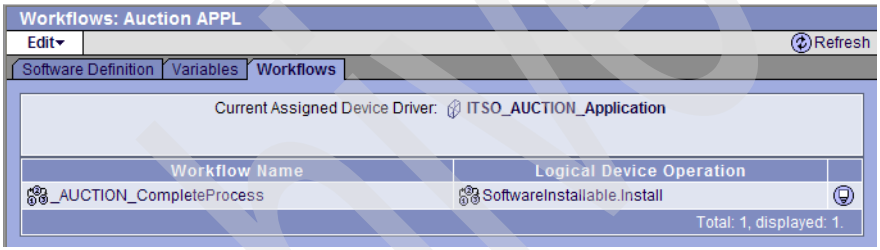

Figure 43 Device driver association

The support team receives the approval for deploying the fixed version of the Auction application. The plan is to deploy the application to the servers dedicated to the Auction WEB Servers application tier defined in IBM Tivoli Provisioning Manager.

On the IBM Tivoli Provisioning Manager GUI, the support person uses the Install Software wizard to perform the installation of the Auction application software package.

During the execution of the Install Software wizard, the Auction APPL software module definition is chosen.

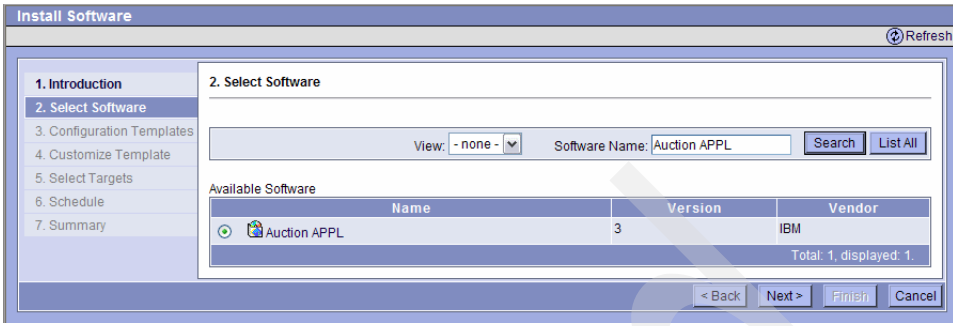

Figure 44 Auction installation- selecting the software module

The wizard presents the Auction APPL configuration template. The support person selects the configuration template AuctionConfiguration. No further modifications is required in the configuration template. It provides all necessary attributes for the installation in progress, as seen in the following figure.

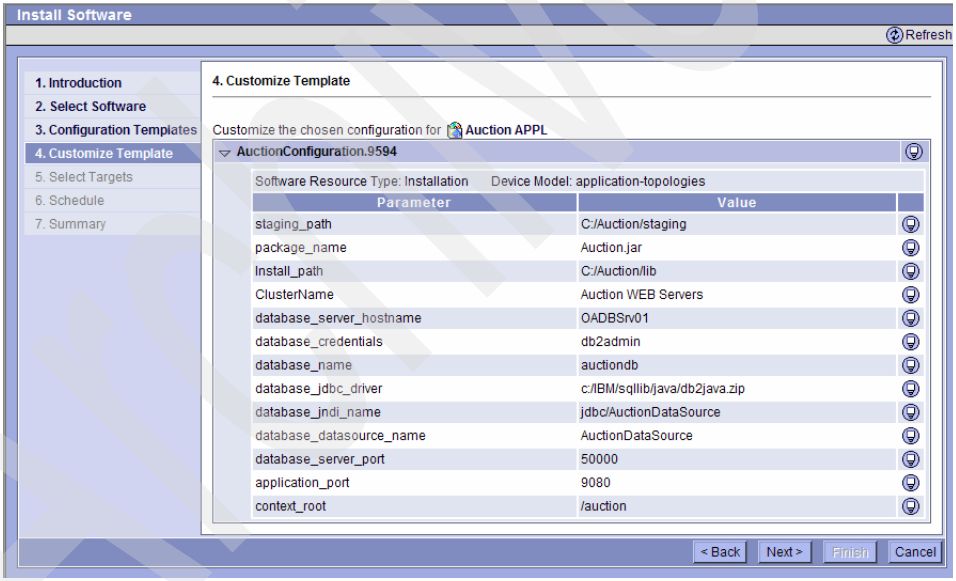

Figure 45 Auction installation- configuration template

The Install Software proceeds by prompting for the targets of this installation. In this case, the fixed version of the Auction application must be installed on all dedicated servers of the Auction WEB Servers application tier, as presented in the following figure.

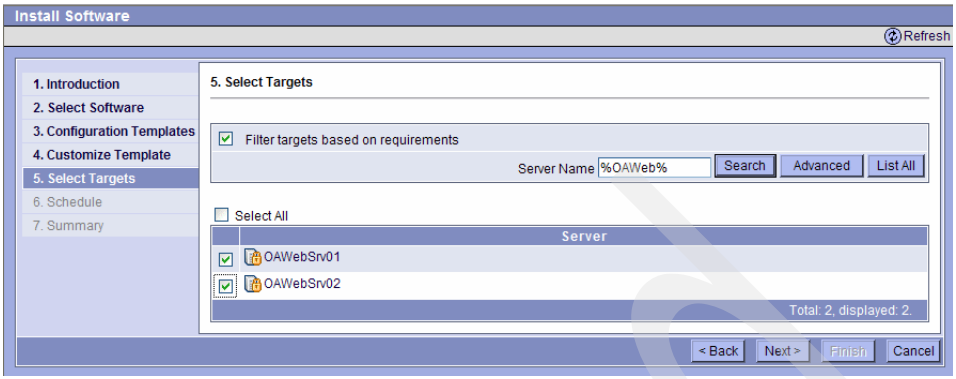

Figure 46 Auction installation- selecting target servers

The next step is to schedule the software installation. As this operation is being performed during the planned maintenance period, the support person chooses to schedule the installation task immediately.

The Install Software wizard then presents a summary panel that presents the details of the installation. Note in Figure 47 that the wizard has already cloned the AuctionConfiguration configuration template (one per target server). These clones will represent how the Auction application will be installed on those targets.

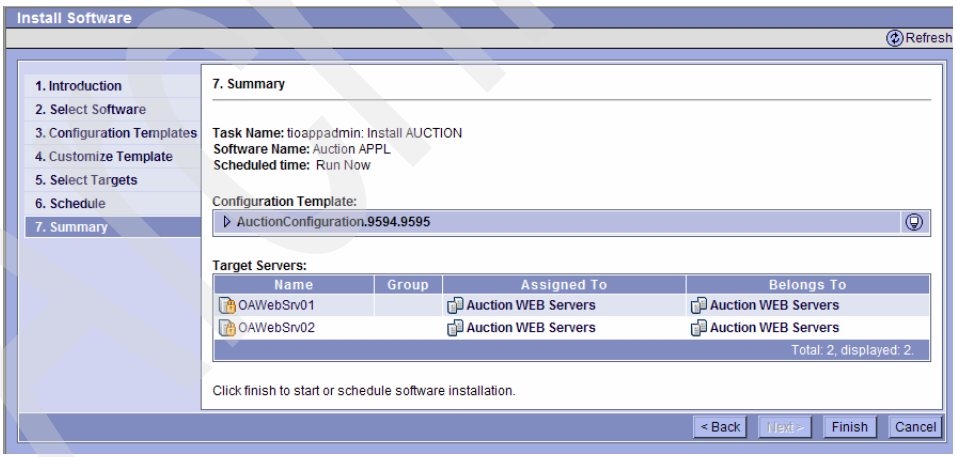

Figure 47 Auction installation- summary panel

Once the support person clicks the Finish button, the task for the installation of the Auction application starts. This is done by IBM Tivoli Provisioning Manager by calling the executing the _AUCTION_CompleteProcess workflow which is

associated with the device driver assigned to the Auction Applet software installable, as presented in Figure 43 on page 70.

As documented in “Workflow design and development” on page 55, the _AUCTION_CompleteProcess implements the SoftwareInstallable.Install logical operation. It executes several other workflows to accomplish the installation of the Auction application.

Once the _AUCTION_CompleteProcess workflows completes execution, the support person verifies the results by checking the Task Details panel, as presented in the following figure.

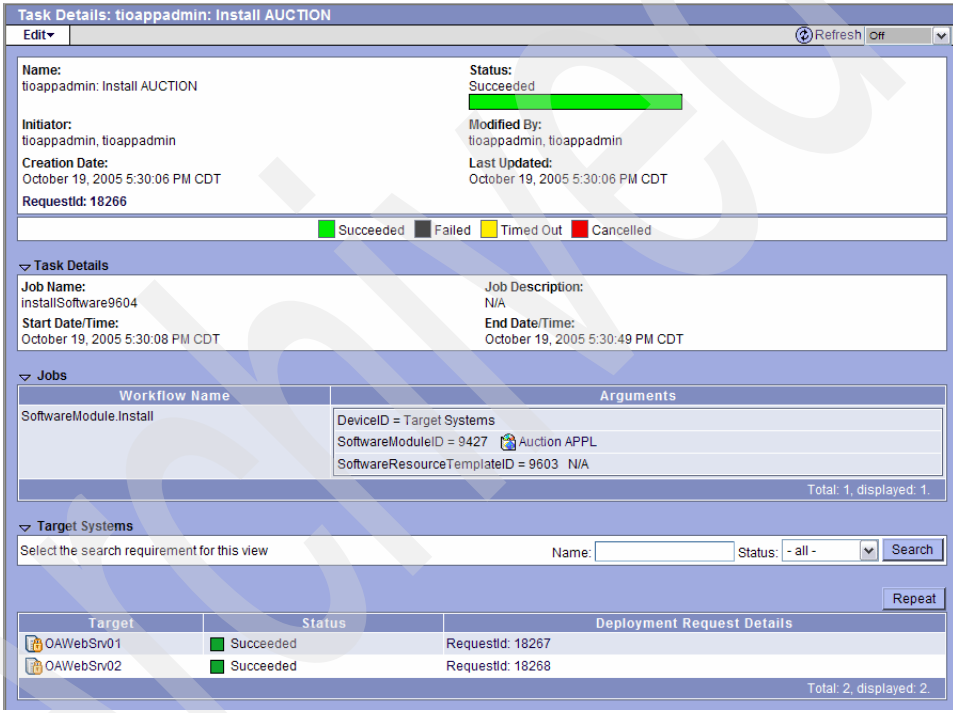

Figure 48 Auction installation- Task Details panel

Details of the workflows execution can be seen by selecting the RequestId performed during the task process.

Conclusion

In this Paper we presented a scenario which traced a resolution of a problem, starting from problem identification in production to resolution in development

and deployment of the fix to production. We covered some of the basic elements of the integration as they are available today between IBM Rational ClearCase and IBM Tivoli Provisioning Manager.

There are other scenarios of this integration that can implemented, such as using IBM Rational ClearQuest with IBM Rational ClearCase and IBM Tivoli Provisioning Manager. There is a more “out-of-box” based integration currently being developed by the engineering teams in both IBM Rational Software and IBM Tivoli Software focusing primarily on compliance based traceability from development to deployment.

Refer to the following Web sites for details on the upcoming releases of:

- IBM Rational ClearCase

  http://www.ibm.com/software/awdtools/clearcase/index.html

- IBM Rational ClearQuest

  http://www.ibm.com/software/awdtools/clearquest/index.html

- IBM Tivoli Provisioning Manager

  http://www.ibm.com/software/tivoli/products/prov-mgr

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Edson Manoel is a Certified IT Specialist at IBM working in the ITSO, Austin Center, in the systems management area. Prior to joining the ITSO, Edson worked in the IBM Software Group, Tivoli Systems, and in IBM Brazil Global Services Organization. He was involved in numerous projects in designing and implementing systems management solutions for IBM clients and Business Partners. Edson holds a bachelor's degree in Applied Mathematics from Universidade de Sao Paulo, Brazil.

Kartik Kanakasabesan is a Consulting IT Specialist with IBM Software Services for Rational based in Toronto, Canada. He has 12 years of experience in the IT field. He holds a degree in Business Management from University of Delhi, India. His areas of expertise include IBM Rational ClearCase, IBM Rational ClearQuest and IBM Rational Team Unifying Platform. He has written extensively on IBM Rational ClearCase and IBM Rational ClearQuest. Kartik's current focus in the field includes integrating IBM Rational and IBM Tivoli products.

Thanks to the following people for their contributions to this project:

Ryan Sappenfield
IBM Rational - Business Development

Morten Moeller
International Technical Support Organization, Austin Center

Paul Tasillo
IBM Rational - Technical Lead Rapid Deploy

Additional material

This Redpaper refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks™ Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/REDP4105

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the Redpaper form number, REDP4105.

Using the Web material

The additional Web material that accompanies this Redpaper includes the following files:

File name	*Description*
REDP4105.zip	Zipped source code and executable files developed for the examples described in the sections of this Redpaper

System requirements for downloading the Web material

The following software configuration is required:

- IBM Tivoli Provisioning Manager V3.1 with fix pack 1 installed
- IBM Rational ClearCase SR5

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

The resulting directory will contain the automation package created during the development of this Redpaper. Follow the instructions provided in the corresponding sections.

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 give 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.

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.

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.

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.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates 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. 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 IBM's application programming interfaces.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. JN9B Building 905, 11501 Burnet Road
  Austin, Texas 78758-3493 U.S.A.

Trademarks

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

ClearCase®	IBM®	Tivoli®
ClearQuest®	Rational®	WebSphere®
DB2 Universal Database™	Redbooks™	
DB2®	Redbooks (logo) ™	

The following terms are trademarks of other companies:

Java, JDBC, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Visual Studio, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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