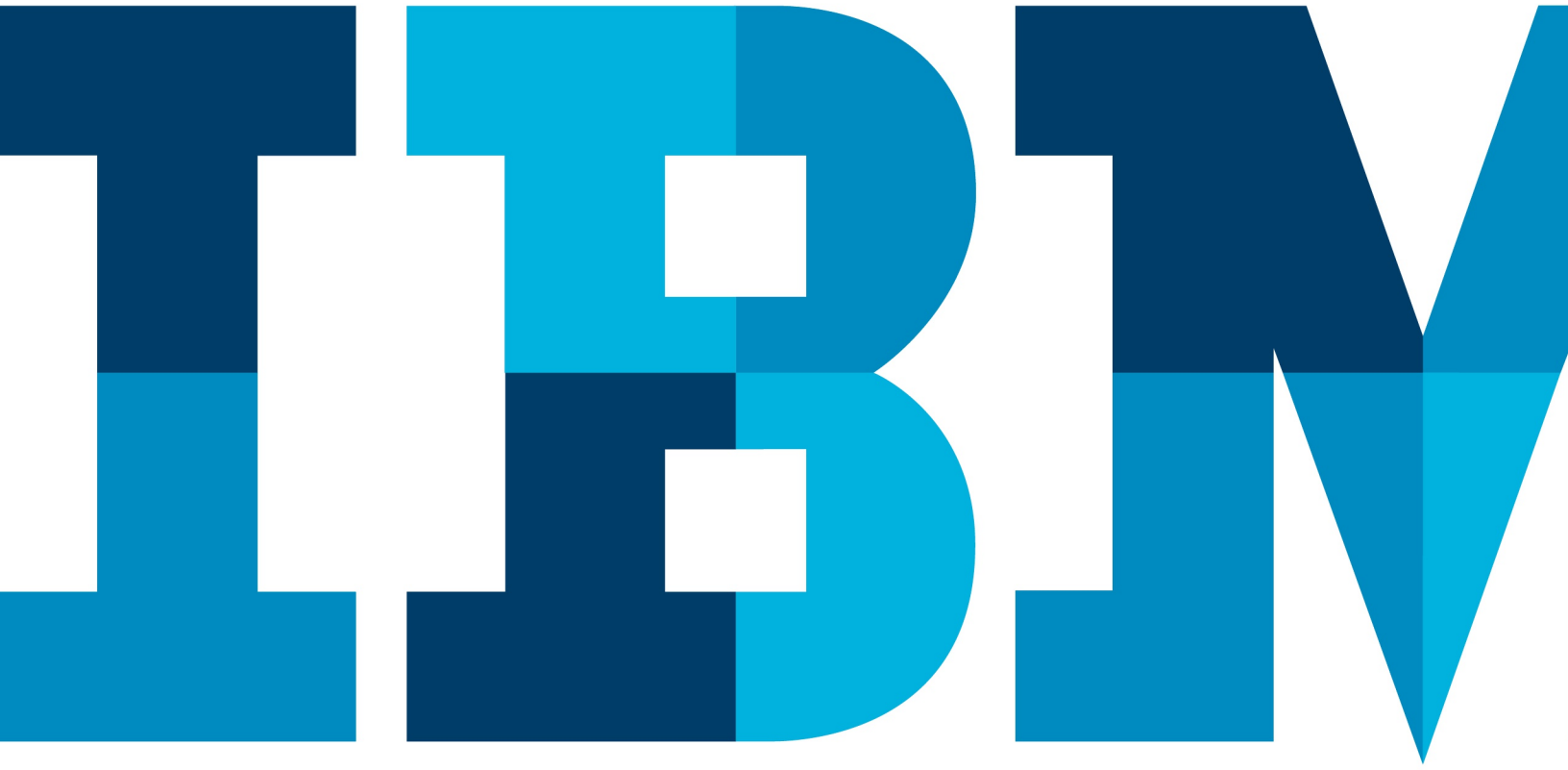
IBM Rational Team Concert Enterprise Extensions Build Administration Workshop

Version 1.0
Lab Exercises



Contents

Overview

This workshop is intended to expose you, as System z build administrator, to many of the tasks that you and your team will need to perform to migrate and maintain your source control and build infrastructure using Rational Team Concert Enterprise Extensions.

At the end of this workshop, you should have an awareness and understanding of the following:

- Upfront planning and decisions involved in preparing for a migration
- Representation of host resources and build steps using RTC system definitions
- Configuration of build processes using RTC build definitions
- Control of development hierarchy (Development->Test->QA->Production) through RTC promotion
- Release of changed applications to Test and Production environments through RTC deployment

Introduction

The labs in this workshop are written with the assumption that you have some familiarity with RTC as a user (e.g., work items and basic SCM functionality) and with Eclipse. You should be able to follow along without this familiarity, but it is recommended to optimize your experience and benefit performing this workshop.

Along with this lab document, you should have received or downloaded the following files:

- eebaw.build.utils.process.ibm.com.zip – Process template for system definitions project area
- eebaw.jkebanking.process.ibm.com.zip – Process template for JKE Banking project area
- EEBAWMortgageApplication.zip – Sample mortgage application
- EEBAWScripts.zip – Scripts for generating system definitions and associating with sample project

You will be using these files throughout the workshop.

Additionally, you should have received or downloaded a presentation containing information to introduce and supplement each lab.

Icons

The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

Workshop setup and preparation

Workshop Scenario:

You are the administrator of the existing software configuration management (SCM) and build infrastructure for host-based applications at a fictitious banking company, JKE Banking. JKE Banking has adopted the Rational Collaborative Lifecycle Management solution to manage and collaborate on all aspects of the software development lifecycle. Your mainframe developers are enjoying the benefits of using Rational Team Concert for work items and planning, and you have made the decision to fully migrate to Rational Team Concert for SCM and build as well. In this workshop, you will learn how to import the host-based mortgage application into source control and create the artifacts necessary to build, promote, and deploy your application.

Workshop Setup:

At this point, you have followed the Installation Instructions document provided with this workshop to configure your environment, or an environment has been provided for you. Following this document, you will generate the project area your JKE Banking developers have been using for work items and planning using a provided template. This one-time setup would typically be performed up front by a project administrator. Note that this is a distinctly separate role from that of EE Build Administrator. The responsibilities of the EE Build Administrator, i.e. source control and build administration for mainframe applications, will be the focus of this workshop.

Workshop scenario materials

This workshop creates and utilizes a simplified portion of the Money that Matters sample application, which is included with the Rational Collaborative Lifecycle Management solution and centers around your fictitious banking company, JKE Banking.

You will prefix all of the artifacts you generate throughout this workshop with EEBAW, for Enterprise Extensions Build Administration Workshop, to avoid any confusion or collision with the artifacts that are generated automatically as part of the Money that Matters sample.

1. Import the provided process templates

Several students sharing an RTC server

This task is to be performed just one time per server. Take that into account if this workshop is being run in a server shared by several students.

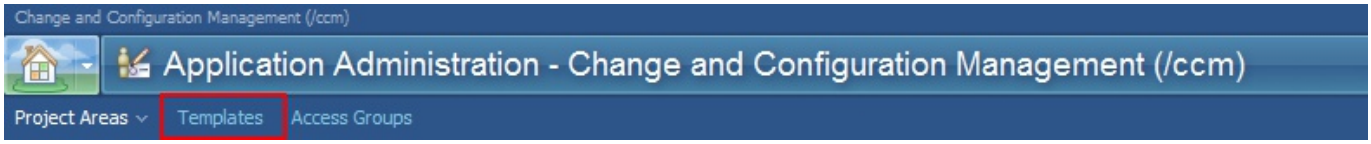
You will be creating two project areas during this workshop. As project area creation and customization are not a focus of this workshop, these steps will be simplified through the use of two customized process templates. The process templates must be imported before they can be used for project area creation.

__1. Use the web client to access your Process Templates.

__a. In your web browser, navigate to https://<*repository_address*>:*port*/ccm/admin

__b. If prompted for credentials, log in with the jazz user ID you created for yourself when you set up the server.

__c. Click the **Templates** link in the top navigation bar.

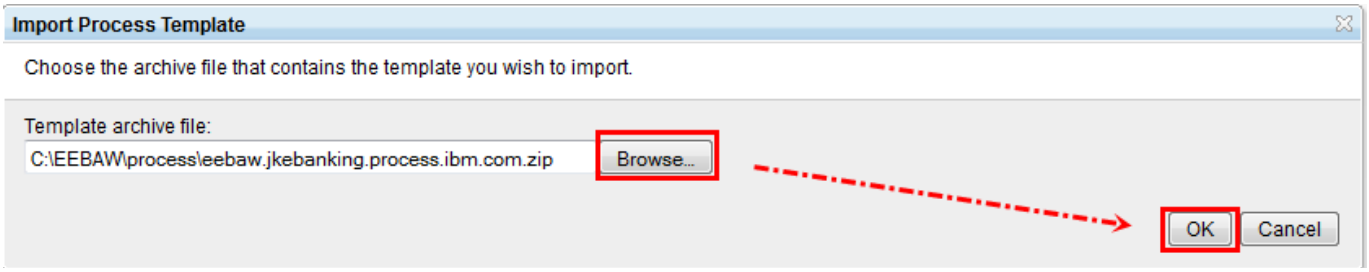


__2. Import the *EEBAW JKE Banking Process* template.

__a. In the upper right-hand corner, click the **Import Template** link.

__b. Use the **Browse...** button to locate the `eebaw.jke.banking.process.ibm.com.zip` file provided with the workshop and select it.

__c. Click **OK** to upload the template.



__3. Import the *EEBAW Build Utils Process* template.

__a. Repeat step 2 above to import the EEBAW Build Utils Process template, found in the `eebaw.build.utils.process.ibm.com.zip` file provided with the workshop.

2. Generate the EEBAW JKE Banking project area

You, as project administrator, will generate JKE Banking's project area that will be used to develop and build the mortgage application.

__1. Create a new project area:

__a. Click **Project Areas** in the top navigation bar and choose **Create > Project Area**.

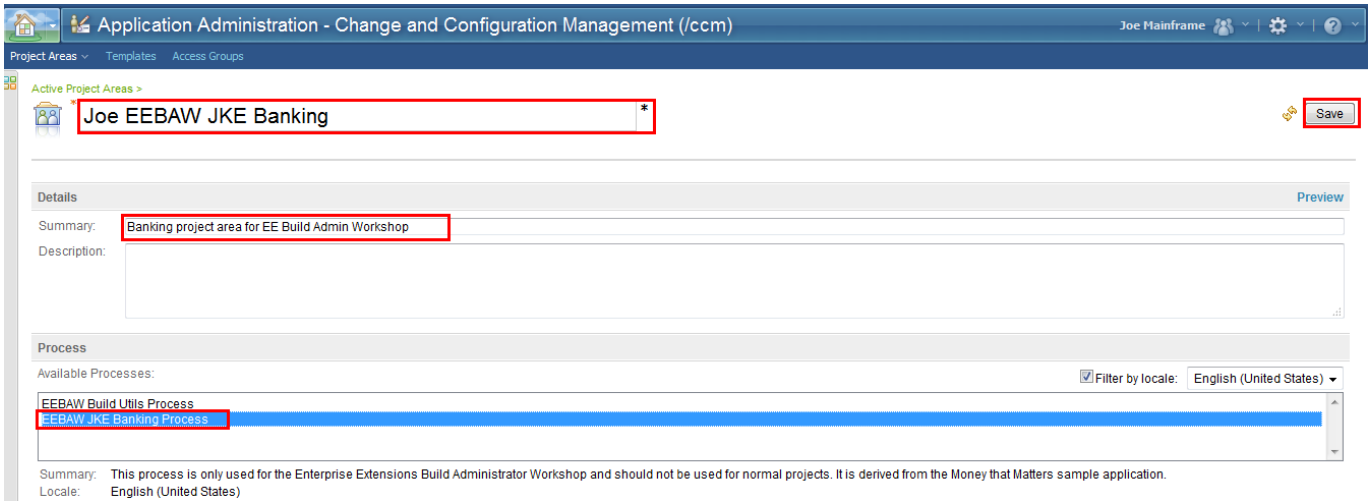
__b. Replace the default text “Project Name” with the name *your_user_ID* `EEBAW JKE Banking.`

Workshop naming convention

Throughout this workshop, you must include your user ID before the EEBAW prefix when creating artifacts if you are sharing your repository server with other workshop students. If you are using a personal sandbox server, your user ID can be omitted.

__c. Enter the following Summary: `Banking project area for EE Build Admin Workshop`

__d. Under Available Processes, choose **EEBAW JKE Banking Process** and click **Save**.



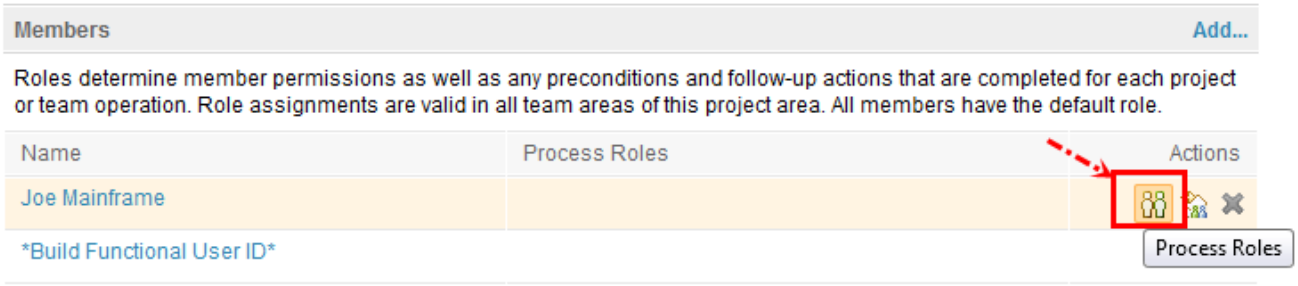
__e. Wait for initialization of the project area to complete. The new project area opens in the Project Area editor.

__2. Add yourself as a member of the project area:

__a. On the Overview page, in the Members section, click **Add...**

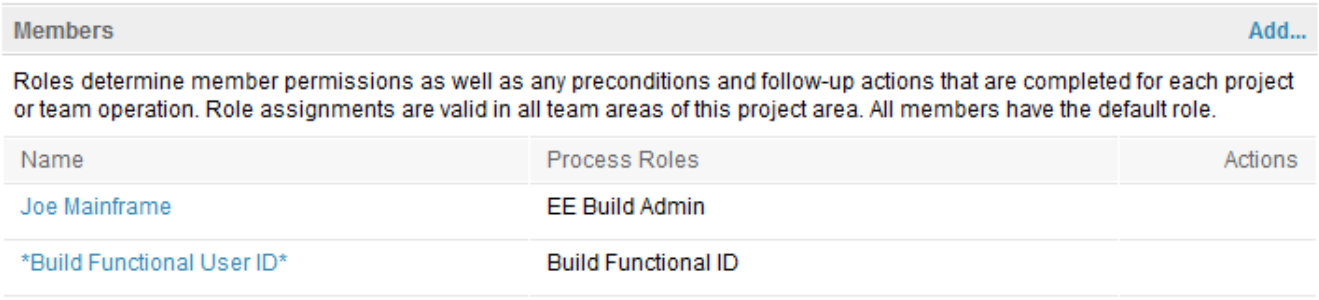
__b. Start typing your name in the Search bar, select your name when it appears, and click **Add.** Without closing the window, look for the builder ID and click **Add and Close**.

__c. Back on the Members section, click the process roles icon for your user.



__d. Select **EE Build Admin** and click the right arrow icon to add the EE Build Admin role to your Selected Roles, then click **OK**.

__e. Repeat the previous step to add the Build Functional ID role to your builder ID user. The final user/roles assignment for the project area will look similar to the following:



Roles Information

Roles determine the permissions given to each member of a project or team area. You will want to create a role in your project area to lock down the creation and modification of several assets that we will see throughout this workshop.

As part of the workshop we provide you with the EE Build Admin role with all the permissions needed for the workshop operations, not just the ones related to EE build artifacts. You will also want a locked down role similar to Build Functional ID to be assigned to your build ID.

__f. Click **Save** to save your changes.

__3. Assign work item categories:

Categories assignment

The provided process template creates the team areas and work item categories that will be used for the workshop scenario by means of Project Area Initialization configured follow-up operations.

However, currently this initialization operation does not support specifying the mapping between team areas and work item categories. You can check the work item Provide a way to keep category / team area association for further information.

__a. Select the **Categories** page in the left hand navigator.

__b. Change an association by hovering over and clicking in the Associated Project/Team Area column for each category, then selecting the desired team and clicking **Associate**. Perform the following mappings:

- "Accounts" category associated with "Account Management" team area
- "Core Components" category associated with "Core Banking" team area
- "Mortgage" category associated with "Mortgage" team area
- Child categories of Mortgage will inherit the team area assignment from the parent.

__i. Click **Save**.

__ii. The final result will look similar to the following:

Actions	Categories	Associated Project/Team Area
	▼ Unassigned <Root Category>	Joe EEBAW JKE Banking [Project Area]
	Accounts	Account Management
	Core Components	Core Banking
	▼ Mortgage	Mortgage
	CICS	Mortgage [inherited]
	Core	Mortgage [inherited]
	User Interface	Mortgage [inherited]

__4. Click **Save** to save any remaining changes to the project area.

Summary

After completing the steps in this document, your environment is ready to complete the workshop scenario.

Lab 1 Planning your Rational Team Concert solution

In this lab you will walk through the investigation necessary to understand your current mainframe SCM environment in order to migrate to Rational Team Concert. You will also create the project area for your enterprise-wide system definitions used to build and deploy your mainframe applications.

JKE Banking is starting the adoption of Rational Team Concert for mainframe development. As an administrator of the mainframe SCM environment you have to gather information from your current environment to map it to the new concepts and constructs you will use in Rational Team Concert for supporting the SCM, build and deploy processes.

In this lab you will:

- Review the basic set of information you would need to consider from your existing environment for migrating to Rational Team Concert. This will help you understand the sample structure to be used in this workshop and use the information as guideline in your real environment.
- Map the information you are gathering to some of the concepts in Rational Team Concert.
- Create a project area to hold the system definitions to be used for building and deploying the applications enterprise wide.

Lessons learned:

- Information to study for a source code solution migration to Rational Team Concert
- How to centrally administer the System Definitions

Migration effort and activities

The purpose of this workshop is to familiarize you with the Rational Team Concert Enterprise Extensions features for building and deploying your mainframe applications. A detailed discussion of the effort and activities involved in a SCM and development environment migration process is out of scope for this workshop.

1.1 Adopting Rational Team Concert

Scenario Recap

JKE Banking mainframe teams have been using Rational Team Concert for work item management and planning for a while.

At this point in time, JKE Banking is interested in adopting the full capabilities that Rational Team Concert offers, moving their mainframe development to use SCM, build and deployment features.

The Mortgage application and the team that owns its development will be the pilot team to migrate to Rational Team Concert SCM and begin using the build features.

You have to plan for the migration of all these processes from the current mainframe development environment, and figure out the layout of the solution in Rational Team Concert.

Theoretical part of the lab

This part of the lab is rather theoretical to introduce you to some of the main discussion points you may face for the solution adoption within your organization.

Hands on will continue in following sections.

1.1.1 Team organization and process: review

JKE Banking mainframe teams are already using work items and planning, so the team structure is already set up. Here we will quickly review the main concepts and information taken into consideration for this team area organization.

Project areas and team areas: What are they for? Project areas are top-level elements in the repository. They represent a software project effort and define the deliverables, team structure and the development process for the context of Rational Team Concert activities for this project area and the managed artifacts. The process for a project area defines the collection of roles, practices, rules, and guidelines that organize and control the flow of work within it.

When designing your solution, think of project areas as a flexible way for organizing your assets and teams. Some project areas can directly represent and map to a software project, while others may be used to represent functional areas within your organization with teams collaborating on a set of related assets.

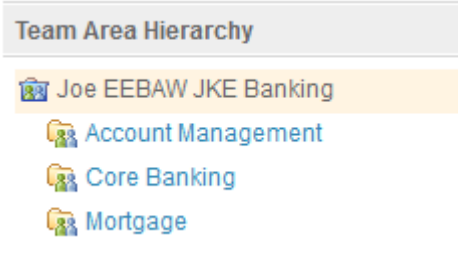
When defining project areas and the process that will govern it, begin using an out of the box process template that best suits your needs and try to tailor only when necessary. As your team continuously works with the solution you will find that adjustments need to be performed in the process, including the need for configuration of work items for new information management needs.

Within a project area you specify the hierarchy of teams working on that project by means of *team areas*. Team areas inherit the process defined for the project area, and they manage team membership, roles assignment and team artifacts. The process can be tailored within team areas to adjust to a team's needs.

Create team areas for groups of people who need to plan and work together. Define a hierarchy of work item categories that functionally map to the areas of work and concerns, so they are meaningful for organizing the work. Keep project leadership at the project area level, while the actual work, team members and artifacts are managed at the team area level.

How is all this realized in the EEBAW JKE Banking sample project area? JKE already went through this exercise when the mainframe teams adopted Rational Team Concert for work items and planning. The resulting structure that you will use in this workshop is based on the following:

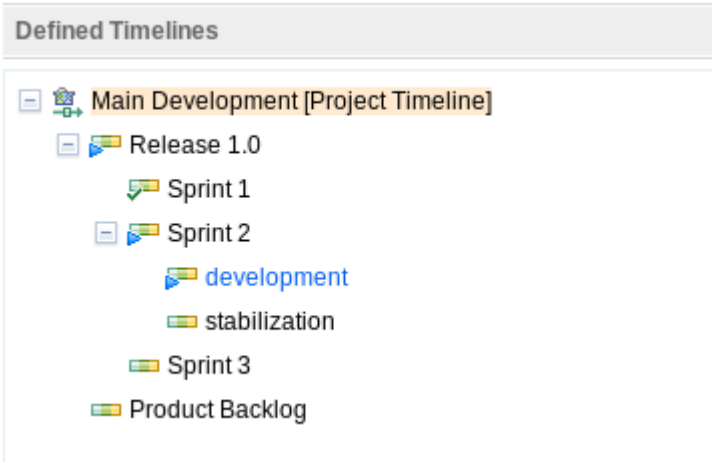
1. The simplified example defines a project area for the JKE banking development. Three team areas exist in the project area for the main development groups.



2. Sample work item categories have been defined. The Mortgage category defines three sub-categories for the basic building parts of the mortgage application components so better categorization of work can be performed.

Categories	Associated Project/Team Area
Unassigned <Root Category>	EEBAW JKE Banking [Project Area]
Accounts	Account Management
Core Components	Core Banking
Mortgage	Mortgage
CICS	Mortgage [inherited]
Core	Mortgage [inherited]
User Interface	Mortgage [inherited]

3. The timeline structure and current iteration looks as follows:



1.1.2 Work Task management

The work task management in Rational Team Concert is performed by means of work items. They are a generalized notion of development task, used to capture information and track the work performed during the development cycle. Different work item types are used to represent different tasks and organize the information, with an associated workflow that defines the “task” lifecycle.

Work item linkage capabilities allow you to express the interrelationship between work items information and with artifacts:

- The linkage between work items allows you to express the interrelationship between different tasks, and to reflect that in your work and in the project plans.
- Linking code changes with work items will provide traceability between the development task and the changes performed while working on it.

All the defining details of work items are configured in the project's process, including the information modification permission for each role. For more information regarding work item customization you can check the jazz.net articles Getting Started with Work Items in Rational Team Concert and Customization of Work Items in Rational Team Concert.

JKE Banking mainframe teams have been using work items for a while for planning purposes. The process the team is already using contains the definition of work item types that will also support tracking the information derived for the initial adoption of the z/OS SCM and build features, so we will assume in the workshop that no further process configuration is needed on this regard.

1.1.3 SCM structure: components and streams

In the process of adopting Rational Team Concert SCM, you have to define how your development assets will be organized, and how code will be shared and flow in the repository. This definition will be usually performed by the development team leaders in collaboration with development architecture and the SCM teams within your enterprise. The definition tasks are beyond the responsibility of the build administrator, although he may contribute to the design with his knowledge of the applications build process.

Components in RTC. Different SCM systems define different structures around which you have to organize your artifacts. In Rational Team Concert, artifacts are organized in *components*, which are the fundamental logical structure for organizing assets in SCM. You have to define how your applications can be logically divided into a set of logical elements called *components*. Components are also used to control access to code, reuse code between teams/applications, or build subsets of the system.

For this componentization effort, several roles within your organization can participate, such as SCM team members, application architects and team leaders. In the end, the work will need to address the following questions:

1. Which logical units make up the applications (the components)? For this study think of putting related artifacts or projects together so components make sense from code reuse, application build operations and team sharing perspective.

2. What are the common source elements used across several applications/modules? Define components to be reused across applications, so they can be maintained by certain teams, or to be shared for all teams within a project area.

3. Components and teams: your development teams will work on a set of components for which they are responsible. When structuring the components along with architectural details bear also in mind the organizational structure that will support it.

You may decide to leverage a source code scanning tool in order to understand the inner structure of the application's code and how to logically split it.

What components will we have in the JKE Banking sample? You will define two components in the JKE Banking project area: Mortgage and Mortgage Common. These components will hold the code for the pilot application and potentially needed core banking code.

Along with the componentization of your applications, you have to consider the *streams* you will need to support your development. A s*tream* is a collection of one or more components used to organize the work of a project, coordinate the integration activities, and capture important configuration points of your source code (promotion levels, releases, integration builds ...). Some of the decisions you will have to make are:

1. How many streams do I need to define? Start with the basic streams to support your integration activities and the levels of code build and promotion performed within your organization.

2. How do you flow your changes? For the different streams you plan for your SCM, you have to define how code version changes will flow between them and what the integration points are.

3. At what levels do you build your code? Code at some stream levels will be used to build, while others are just for configuration traceability.

Various aspects of your current workflow will influence your final stream strategy, such as whether you are doing release maintenance, how you handle emergency fixes, whether you require different integration levels for different teams, etc. But when planning the first adoption of the new SCM solution, you just want to consider the structure for the basic phases of your development cycle. As teams work in RTC and get familiar with how the SCM works, you can define new streams that will support additional needs. You may discover through this process that aspects and portions of your current workflow are no longer ideal or necessary with your RTC solution, and should not be replicated.

What streams will we use in JKE Banking? For the migration of the pilot application and team, you will create three streams: EEBAW JKE Development Stream, EEBAW JKE QA Stream and EEBAW JKE Production. These are the basic streams that will be needed for integrating, building and deploying the mortgage application. Note that future applications and teams adopting these features of Rational Team Concert within JKE Banking may require additional streams, but at this point you will start using just the basic needed blocks.

1.1.4 How you build your applications

Adopting the new SCM solution is not just about organizing code, but also supporting the processes to build that code and deploy the results to the runtime systems. Rational Team Concert Enterprise Extensions provide capabilities for building, promoting, packaging and deploying your code; hence a deep understanding of how these processes are currently implemented in your system is needed to migrate them to Rational Team Concert:

1. Understanding the build process of your applications: what are different technologies in use for building your applications and how do you build them.

2. “Stages” of your source code and where do you build applications: you already reviewed this information when defining the stream structure. You will also use what you learned for defining the promotion process in Rational Team Concert, and the streams in which builds are performed.

3. Understanding how your applications are deployed: what are all the details of your target deployment locations and your runtime environments

Rational Team Concert provides a set of features to support all these operations for your enterprise environment. A brief description of these features follows:

- System Definitions: contain supporting information for the build features

  - Language Definitions: define the steps to be performed for building a type of file, and what scanner will gather source code data for that type. Each file to be built or scanned must be associated with a language definition. The build steps of a language definition are specified by means of translators.

  - Translators: define a build operation/step that will be performed on a file during a build, such as a compile or link-edit. Translators contain all the necessary details for the operation, such as the input and output data sets, syslib specification, etc.

  - Data Set Definitions: provide location information for all the data sets used for your build process. This includes input/output data sets, temporary data sets, and location information for the compilers or other types of programs to be used in your build process.

- Build specifications:

  - Rational Team Concert builds code using the constructs of Build Definition (build script and build properties information to use) and Build Engine (the machine where code is built/deployed; e.g. your mainframe systems).

  - Additionally, for z/OS development, RTC provides a feature-rich special kind of build definition called z/OS Dependency Build. This type of build definition will allow you to just build assets that have been modified, along with the dependent elements.

- Promotion: used to move source code and the resulting build artifacts in the promotion hierarchy.

- Packaging and Deployment: features that will allow you to create a package from the build results and move them to the target runtime environments.

Just as you did when defining your stream strategy, you may find that parts of your build process are inefficient, overly complex, and unnecessary. You should aim to simplify your build process as you migrate to RTC, rather than simply reproduce it.

In the remainder of this workshop you will walk through all of these features in detail: as part of the JKE Banking enterprise you will migrate the mortgage application as the pilot application for adopting Rational Team Concert for SCM, build and deployment.

1.1.5 Solution adoption

You have reviewed the main information that needs to be understood for migrating from your current mainframe development environment to Rational Team Concert. Now you have all the information in place and a draft design of how you plan JKE Banking to work in the new environment. How can the solution be adopted? Some of the usual steps you think of are the following:

1. Review the current process for new needs: for a team that has been using work items and planning, the new features to adopt in Rational Team Concert will usually result in process changes needed to support them. This will typically include:

    - Roles permissions: adjust the developer roles permissions to the new SCM and build operations that they will be performing.

    - Need for new roles: as part of the new features your team is adopting, need for new roles typically arise. The process your team is already following may already contain some out-of-the-box roles that, with some permissions review, would fit your needs; or you may need to define completely new roles in the process. In this workshop the provided process already considers a dedicated role for you, the Enterprise Build Administrator.

    - Work Task management: the activities that the developers will perform related with SCM and build may require that new work item types are created or some customization on the existing types is required to capture all the desired information and traceability.

    - Additional process configuration: this may include configuring the promotion, packaging and deployment summary work items; customizing the team dashboards to include SCM and build related information or configuring additional preconditions or follow-up actions for the new operations. As previously discussed, try to identify and configure the fundamental configuration needed; solution usage and its tracking will reveal future further configuration needs.

2. Define the naming convention for all the elements you plan to create in Rational Team Concert. This should be documented and followed, so the solution is easily maintained and understood by the team members.

3. Identify a pilot application (or a set of them): the pilot application should be relevant in terms of technology and processes. Ideally, the team for the pilot application will just be focused on that (or minimum other work), so they don't have to duplicate efforts in different tools.

4. Define the code migration process: you have to identify where in your system is the source code, and how to extract and import it to the target SCM structure. Some reorganization of the source PDSs may be needed. You will further review in this workshop how to import your code into Rational Team Concert.

5. Implement the elements for supporting your solution design for the SCM and build processes according to the information gathered.

6. Migrate the pilot applications/teams.

7. Validate and adjust the implementation.

8. Plan a gradual adoption for the rest of the teams and applications.

These are just general steps to consider that you will have to tailor and extend with others for your particular needs.

From this point forward, we will perform the migration operations for the pilot application team of Mortgage Application, using the components structure described in this part of the lab.

1.2 System Definitions creation

You have already gathered all the information you need, have a clear idea of what elements are generally needed to build your applications and how you want to organize your code and build processes in Rational Team Concert. In this part of the lab, you will create a project area and the system definitions that are to be widely used in JKE Banking enterprise.

System Definitions project area?

The System Definitions elements are repository wide. Using a project area for the system definitions that are generally used by all the projects eases administration and maintainability of your solution. You must make sure that this project area is accessible from the project areas that will utilize these common system definitions (grant read permissions).

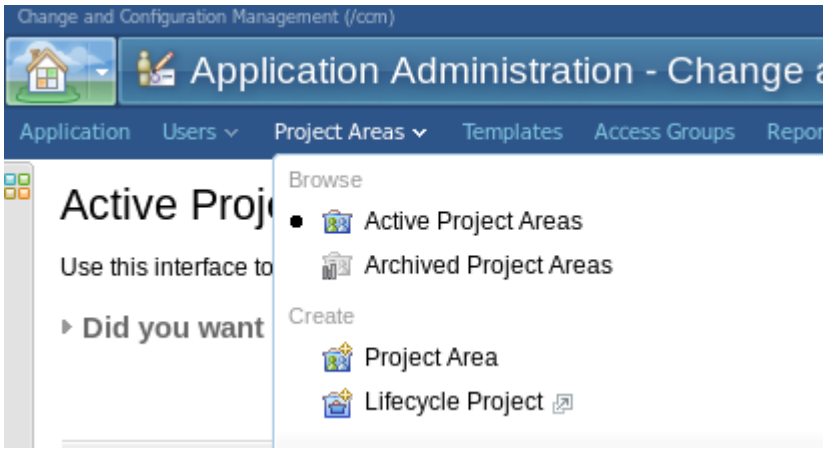
1.2.1 Project area for managing the system definitions

__1. Use the web client to create the project area:

__a. In your web browser, navigate to *https://<repository_address>:port/ccm/admin*

__b. If prompted for credentials, log in with the jazz user ID you created for yourself when you set up the server which has JazzProjectAdmin role assigned.

__c. Click **Project Areas** in the top navigation bar and choose **Create > Project Area**.



__d. Replace the default text “Project Name” with the name *your_user_ID* `EEBAW JKE Common Build Utilities`

Project Area name

Remember that including your user ID as part of the project are name is just advised in case you are sharing your repository server with other workshop students, otherwise it can be omitted.

__e. Enter the following Summary: `Project Area for JKE Banking Build Utils`

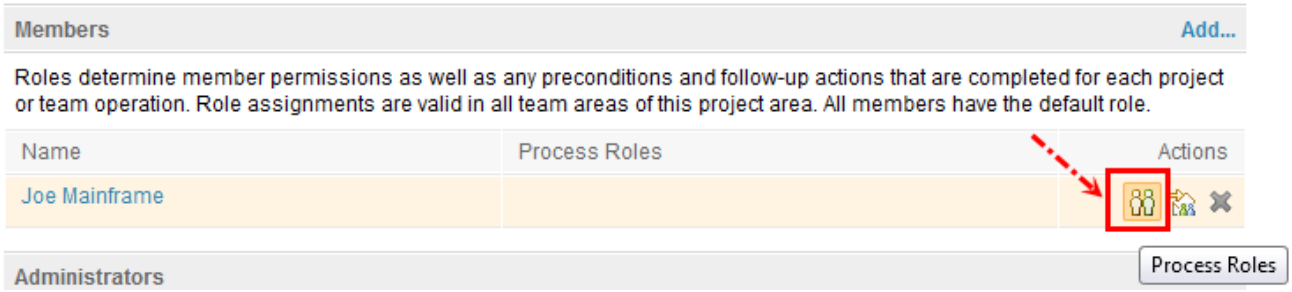
__f. Under Available Processes, choose **EEBAW Build Utils Process** and click **Save**.

__2. Add yourself as a member of the project area:

__a. On the Overview page, in the Members section, click **Add...**

__b. Start typing your name in the Search bar, select your name when it appears, and click **Add and Close**.

__c. Click the process roles icon for your user:



__d. On the Select Roles page, select **Team Member** and click the right arrow icon to add the role to your Selected Roles.

__e. Repeat the previous step for the **EE Build Admin** role.

__f. Click **Finish**

__g. Click **Save** to save your changes.

1.2.2 Generate system definitions

Next, you will use the system definitions generator to automatically create the system definitions necessary to load and build your source on z/OS.

All the details from the beginning?

You don't need to figure out all the system definitions details for your enterprise for the initial adoption of the Rational Team Concert solution. The analysis of your current build processes will reveal however the common enterprise-wide element definitions.

In this part of the lab you will understand how to bulk create that initial information. In later labs you will learn how to manually create new system definitions while you discover the need for them.

__1. Create a repository connection to your workshop server:

__a. Start your Rational Team Concert client

__b. For a newly created workspace the Welcome page will open. Close it and open the **Work Items** perspective if not already open.

__c. In the Team Artifacts view, click the **Create Repository Connection** link

Connection in existing workspace

It is recommended to use a new eclipse workspace for the workshop. If you are reusing a workspace with an existing connection, you will have to instead right click the Repository Connections folder and select New > Jazz Repository Connection to create your connection.

__a. Enter the details for the workshop server connection and select **OK**. If prompted, accept any certificate for the connection.

__2. Import the workshop scripts project:

__a. In your Rational Team Concert eclipse client, switch to the **Resource** perspective

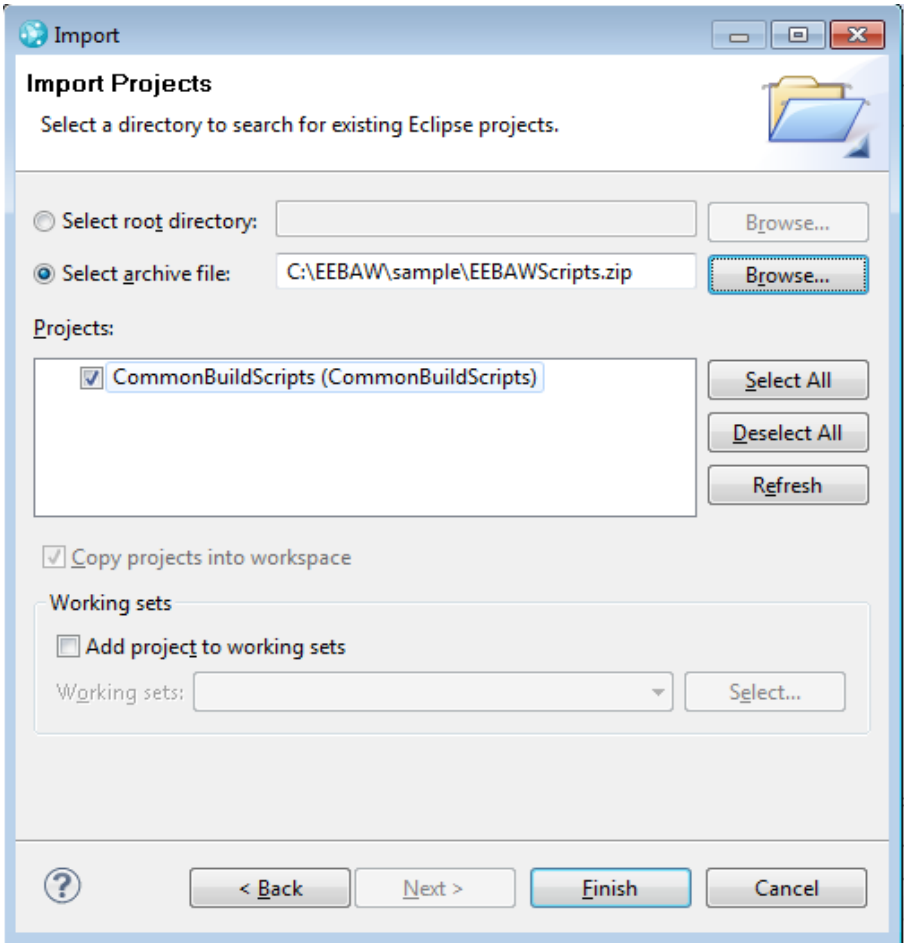
__b. Select **File > Import**

__c. Select **General > Existing Projects into Workspace** and then **Next**

__d. Click on the **Select archive file** option and then **Browse**

__e. Locate the `EEBAWScripts.zip` file from the workshop materials, highlight it and select **Open**.

__f. Click **Finish** to import it into your workspace.

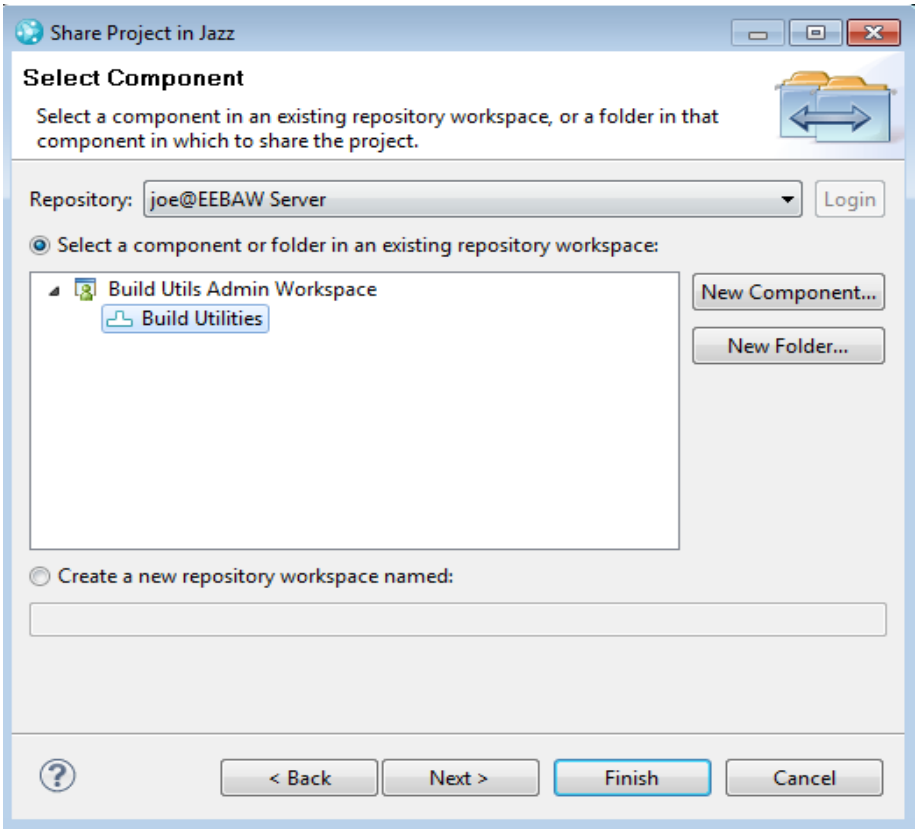


__3. Share the CommonBuildScripts project;

__a. In the Project Explorer view, right click the CommonBuildScripts project and select **Team > Share Project**

__b. Select **Jazz Source Control** for the repository type and then click **Next**

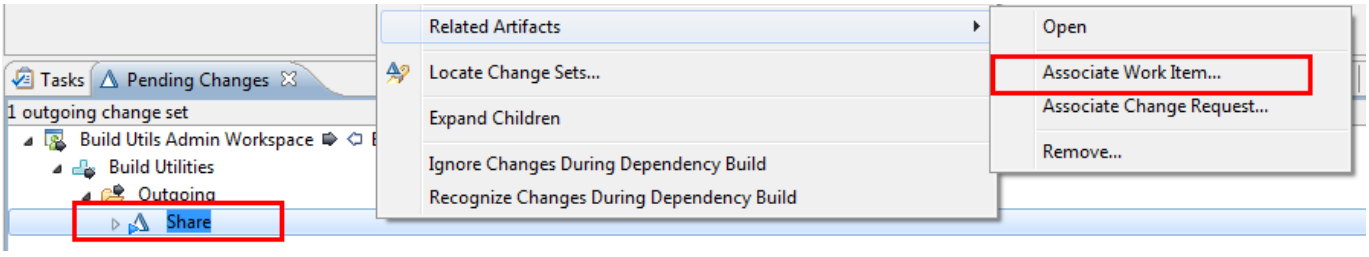
__c. Select the **Build Utilities** component under the automatically created repository workspace called *Build Utils Admin Workspace* and then click **Finish**.



You should now see the project share in the workspace flowing with the Build Common Utils Stream, in the Pending Changes view.

__d. Associate the change set with a work item and deliver changes:

__i. Right-click the outgoing change set called Share and select **Related Artifacts > Associate Work Item**.



__ii. On the Associate Work Item page, select the `Share build scripts with Jazz Source Control` work item. Start typing this title in the search field until the work item appears, then select the work item. Click **OK**.

__iii. Right-click the outgoing change set and select **Deliver**.

__4. Expand the **CommonBuildScripts** project and double-click `setupCommonSystemDefs.xml` file to open it in the Ant editor.

Systems Definitions Generator

The generator is based on a series of custom Ant tasks provided by Rational Team Concert Enterprise Extensions features to programmatically create language definitions, translators and data set definitions.

The Ant script that you will run in this lab is tailored for the workshop sample scenario.

Find more information regarding the generator in Generating z/OS build metadata automatically using the system definitions generator or about the lab script in the Appendix section.

__a. Look in the file for the properties that define the prefixes for your compilers, DB2 libraries, etc. You can use **Edit > Find/Replace** menu or **Ctrl+F** to search in the file

```
<!-- Install location of compilers, etc. Modify to match your z/OS installation-->
<property name="prefix.asm" value="HLA" />
<property name="prefix.cobol" value="IGY420" />
<property name="prefix.cics" value="DFH420.CICS" />
<property name="prefix.db2" value="DSNA10" />
<property name="prefix.rdz" value="FEK850" />
```

Update these locations as necessary to match your runtime environment. Be sure to save the file if you make any changes.

__b. Look as well for the resource.def.prefix property configured. All system definitions generated will have the prefix specified here, **EEBAW**. If you are working in the same repository as another student performing this lab, add your name to the prefix as well to avoid collisions (e.g., Joe EEBAW). System definition names are unique across the repository.

```
<property name="resource.def.suffix" value="" />
<property name="resource.def.prefix" value="Joe EEBAW " />
```

__c. Continue to scroll through the contents of the Ant script, and see that there are tasks for creating the three different types of system definitions for z/OS: data set definitions, translators, and language definitions. The appendix section of this lab has further information regarding the structure and contents of this script file.

__d. Close the editor.

Delivering the script prefix change

If you changed the script during steps “a” or “b” above, you will have created an unchecked in change. You should then perform the following steps:

1. Open the Pending Changes view.
2. Right-click the Unresolved folder and select Check-in and Deliver.
3. Re-use the same work item you used previously to deliver the change.

__5. Right-click the `setupCommonSystemDefs.xml` file and select **Run As > Ant Build...** (note the ellipsis), to open the Edit Configuration wizard:

“Ant Build...” instead of “Ant Build”

Clicking Run As > Ant Build... (with ellipsis at the end) will open the Ant launch configuration editor, where clicking “Ant Build” will directly try to execute the Ant script calling the default target.

__a. On the Targets tab, confirm that the “all” target is selected to execute.

__b. On the JRE tab, make sure that **Run in the same JRE as the workspace** is selected.

__c. Click **Run** to close the wizard and execute the Ant script.

__6. Observe the execution of the system definitions generator in the Console view. The tool should complete with a message like the following:

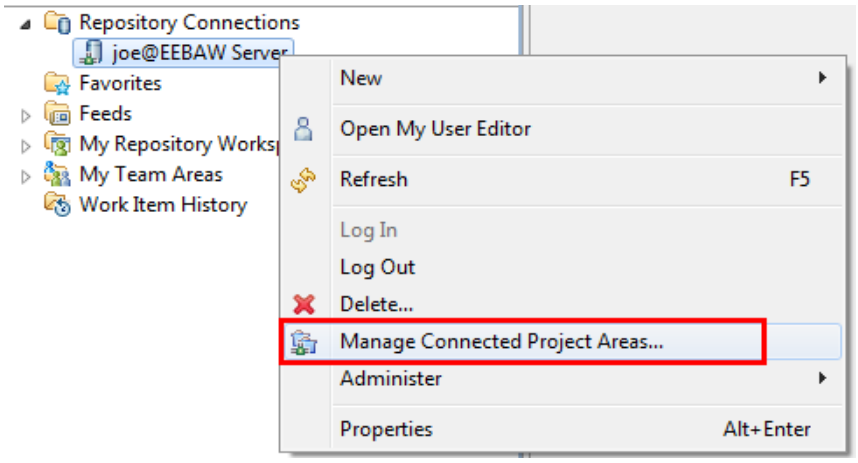
```
all:
BUILD SUCCESSFUL
Total time: 9 seconds
```

1.3 Explore the data set definitions

As previously introduced in this lab, a data set definition is needed to represent every MVS data set involved in an RTC build process. This includes the data sets where the source will be loaded from the repository, the data sets where the build outputs will be created, any necessary temporary data sets, and even the compiler and link-edit modules. You will now explore the data set definitions created by the system definitions generator in the previous lab section.

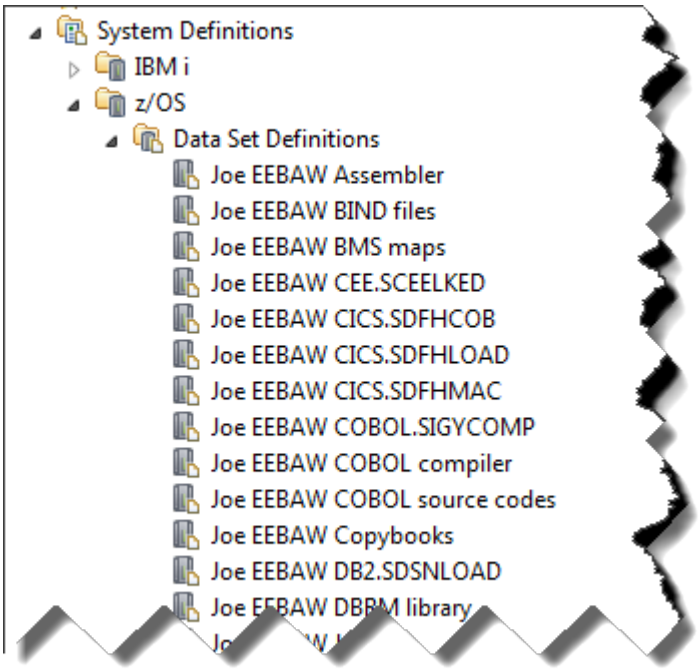
__1. Switch to the Work Items perspective if not already there.

__2. In the Team Artifacts view, right click your workshop server repository connection and click **Manage Connected Project Areas.**



__3. Click **Select All** and then click **Finish**. This will give you access to the project areas you are a member of.

__4. Expand ***`your_user_ID`*** **EEBAW JKE Common Build Utilities > Enterprise Extensions > System Definitions > z/OS > Data Set Definitions**.



__5. Double-click the EEBAW COBOL source codes data set definition to open it in the Data Set Definition editor and take note of the following:

- The **Destination data set for a zFolder** usage indicates that when the build executes, this data set will be created on the host (if it does not already exist) and source code members found in any zFolder associated with this data set definition will be loaded.
- The **Data set name** value and **Add data set prefix from build definition to data set name** options tell us that the data set created will be named *YOUR.BUILD.HLQ*.COBOL, where *YOUR.BUILD.HLQ* is the high-level qualifier you specify in your build definition.
- The **Data Set Characteristics** are the physical attributes that will be used to create the data set on the build machine if the data set does not already exist.

Hold on... what are zFolders?

Rational Team Concert uses a special eclipse project called zComponent project. Source code within a zComponent project is organized in zFolders, which you associate with data set definitions to specify the z/OS data sets that will be used for loading the source members at build time.

In the next lab you will review in detail the structure of this specialized eclipse project.

__6. Double-click any of the “EEBAW CICS.*” data set definitions to open it in the Data Set Definition editor. Note the usage type of Existing data set used for build and that the Data Set Characteristics are disabled. This data set definition tells the build process where the CICS libraries are located.

__7. Double-click the EEBAW COBOL compiler data set definition. Notice that here we are specifying a member name. This data set definition allows the build process to locate the actual compiler module.

__8. Continue to open data set definitions of interest and explore their contents. When you are finished, close any of the open data set definition editors.

1.4 Create a data set definition

You will now create an additional data set definition that was not included in the setup script to be used for building the mortgage application. This data set definition will reference an existing data set on the host, SYS1.CSSLIB, that provides stubs for IBM's callable services.

__1. In the Team Artifacts view, expand the following: ***your_user_ID*** **EEBAW JKE Common Build Utilities > Enterprise Extensions > System Definitions > z/OS**.

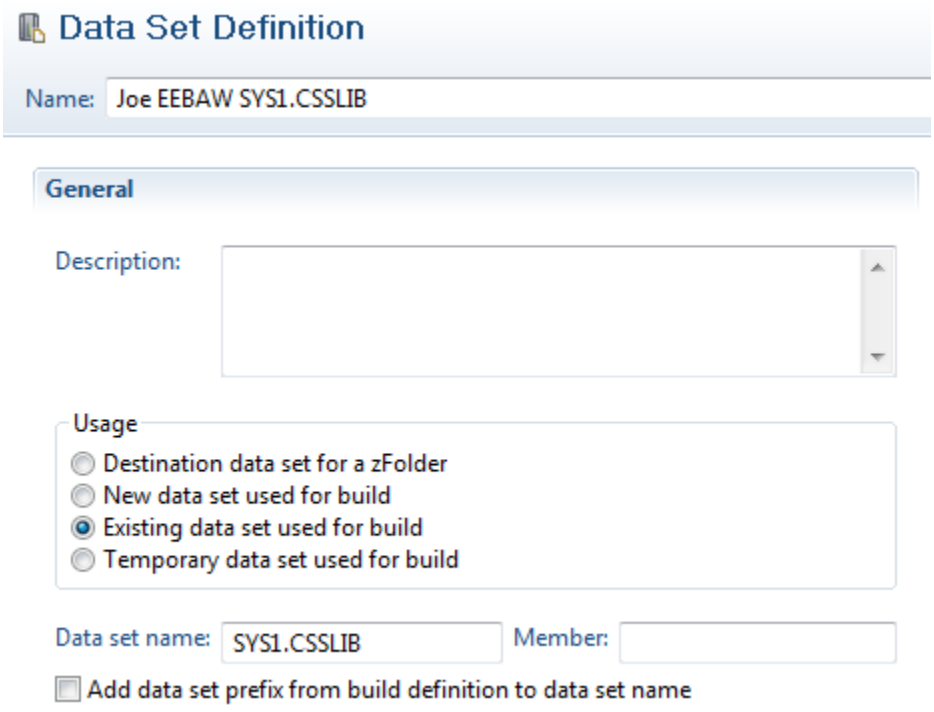
__2. Right-click Data Set Definitions and choose **New Data Set Definition**. The Data Set Definition Editor opens:

__a. In the Name field, type `your_user_ID EEBAW SYS1.CSSLIB`, being sure to add an additional prefix to match your system definition prefixes specified in step *2.2.2.3.b* above.

__b. For Usage, select **Existing data set used for build**.

__c. In the Data set name field, type `SYS1.CSSLIB`.

__d. **Save** the editor contents using **Ctrl-S** and close the new data set definition.



1.5 Summary

In this lab you have reviewed some of the basic information and concepts that you have to take into account when you plan the Rational Team Concert SCM and build solution adoption for z/OS application development.

As part of the adoption of the solution by JKE Banking enterprise, you have created a project area to hold the enterprise-wide definitions to be used, and bulk generated the system definitions that resulted from analyzing the current z/OS development environment at JKE Banking. You finally reviewed the data set definitions information and manually created a data set definition that you will need in JKE environment.

1.6 Appendix: System Definitions Generator lab script review

In this appendix section of the lab, we will briefly review the main details of the system definitions generator ANT script that you used in this lab. Note that complete information regarding the generator feature and the ANT tasks available can be found in this Information Center link.

The structure of the script file is as follows:

- Properties: all the values used in the script are property based and defined within the script. This eases future modification for tailoring for your purposes. In addition, the values being used define a prefix and a suffix that can be easily modified to easily change the name and avoid collisions:

```
35      <property name="resource.def.suffix" value="" />
36      <property name="resource.def.prefix" value="EEBAW " />
```

```
46  <property name="dsdef.bind.source" value="${resource.def.prefix}BIND files${resource.def.suffix}" />
```

  - The properties for the system definitions are structured in three blocks, one for each of the system definitions types: language definitions, translators and data set definitions; for easy location and customization of the values.

- There are different targets for the creation of the different system definitions:

  - There is a “common” target for each kind of system definition to create the definitions with the usual values: target name="translators.common", target name="dsdefs.common" and target name="langdefs.common"

  - For each of the system definitions, there is a target with name suffix “workshop” (i.e. target name="translators.workshop"), for creating elements definition needed for the workshop.

- *Special purpose* targets in the Ant file:

  - Targets for bind operations: there are targets for creating language definitions and translators for performing a DB2 bind. These targets are called: “translators.bind_1”, “langdefs.bind_1”, “langdefs.bind_2” and “translators.bind_2”. These system definitions are based on the article called Performing a DB2 bind with Rational Team Concert 4.0, where the basic options covered are:

    - Option 1 (using system definitions created by “translators.bind_1”and “langdefs.bind_1” targets): the bind is performed in-line for every needed file as part of the build process; as a translator performs the build for every file with the associated language definition.

    - Option 2 (using system definitions created by “translators.bind_2”and “langdefs.bind_2” targets): all binds occur at the end after all programs have been compiled and link-edited.

Check the referred article for further instructions.

- Targets with special debugging options: there are a couple of targets for creating translators called "translators.error_feedback" and "translators.no_error_feedback" which allow you to include compilation errors information from RDz in your build results. Check the following Information Center topic for further information.

Lab 2 Sharing your source members in Rational Team Concert

In this lab, you will add your mainframe COBOL application to the Rational Team Concert repository for source control and associate your source with the system definitions you created in the previous lab.

To take advantage of the source control and build capabilities for z/OS provided by Rational Team Concert, you must organize your source in a specialized type of eclipse project known as a zComponent project. We will explore this project structure in this lab. An SCM command line tool, zimport, is available for importing your source from your MVS data sets into RTC in the proper format automatically. A typical migration would involve organizing your source data sets on the host, creating a mapping file, and running this mass import tool. For simplicity in this workshop, you will instead import existing zComponent projects into your eclipse workspace and share them to the repository.

The steps we will follow are:

1. Create a stream and component structure.
2. Import the sample mortgage application and explore the zComponent project structure.
3. Share the zComponent projects in the Rational Team Concert repository.
4. Associate the source with system definitions and deliver the projects.

Lessons learned:

- stream and component creation
- zComponent project structure
- associating system definitions with zComponent project artifacts

2.1 Create the Production stream

You determined during your initial migration investigation during the last lab that you will organize your source into two components: Mortgage and Mortgage Common. You determined that you initially will need three streams for integrating changes, building, testing, and deploying the mortgage application. You will create the components and your highest level (Production) stream now, and will create the remaining streams in the next lab when you configure your project for promotion.

__1. If it's not still open from Lab 1, start your Rational Team Concert client and open the Work Items perspective.

__2. Create the Production stream.

__a. Within the `your_user_ID EEBAW JKE Banking` project area, right-click the **Source Control** node and click **New > Stream.**

Project area switch!
Notice that we are now working in the EEBAW JKE Banking project area, not the EEBAW JKE Common Build Utilities project area we worked with in Lab 1.

__b. Enter the name `your_user_id EEBAW JKE Mortgage Production Stream` and click **Save**.

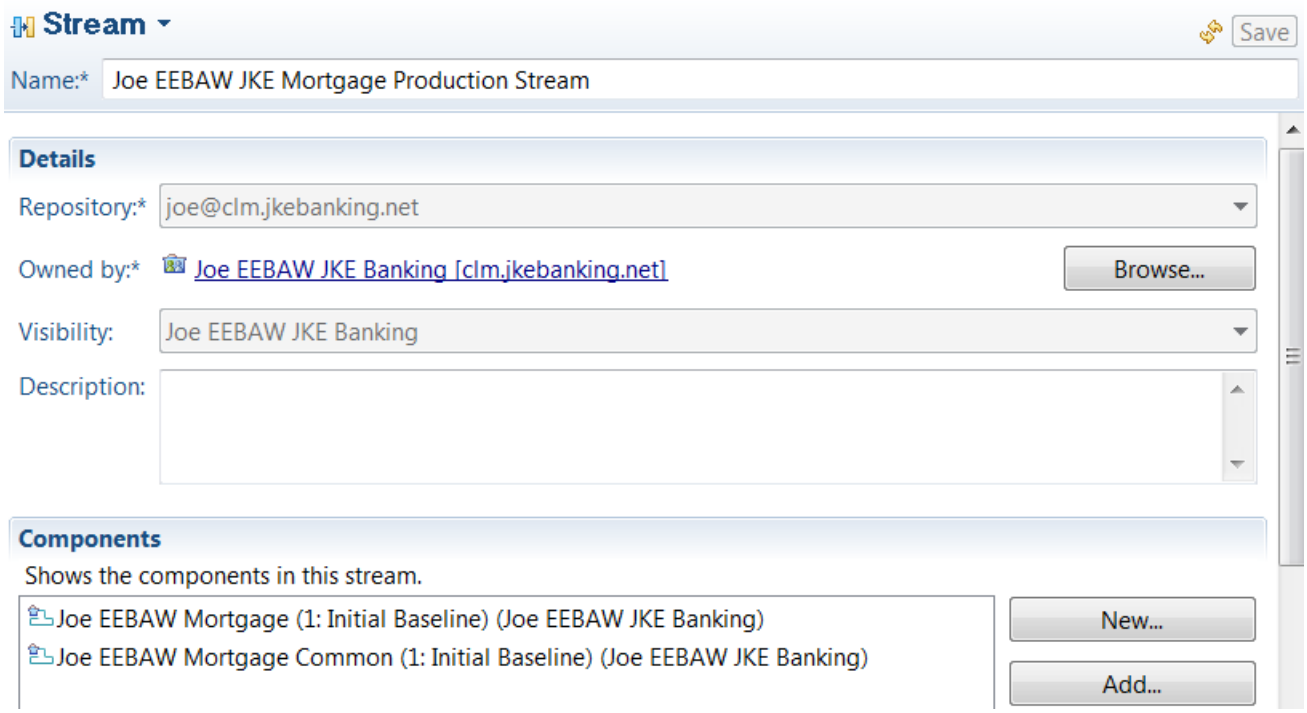
__3. Create the Mortgage Common component.

__a. Still in the stream editor, in the Components section click **New...**.

__b. Type the name `your_user_id EEBAW Mortgage Common` and click **OK**. Ctrl-S to save.

__4. Repeat step 3 to create the mortgage component called `your_user_id EEBAW Mortgage`.

__5. Your stream should now look like the following:



__6. **Save** any changes and close the stream editor.

Stream and component ownership

The stream and components you just created were owned by the project area by default. You could also assign ownership of a stream or component to a team area or individual. This ownership impacts who has read and write access to your source code, and should be considered during the course of your migration. Refer to Controlling access to source control in Rational Team Concert for detailed scenario information.

2.2 Import sample mortgage application

You will import the sample mortgage application as a set of existing zComponent projects.

zImport mass import tool

In a real migration, you would use the zimport scm command line mass import tool to share your source from MVS data sets into the repository in the proper zComponent project structure. Visit the Information Center for more details on zimport.

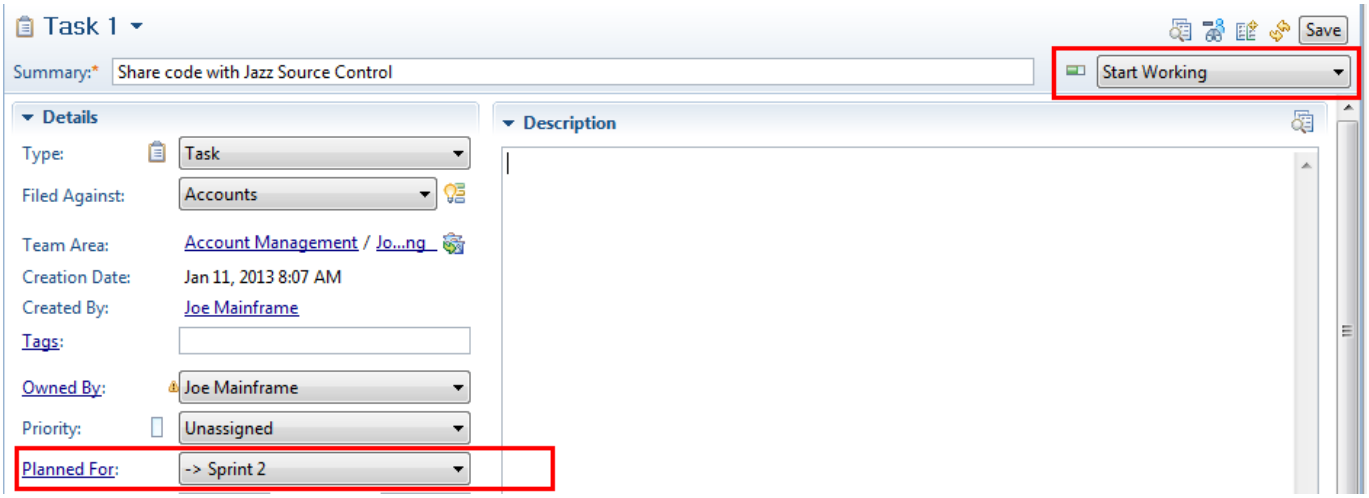
__1. Indicate that you are working on sharing your source in RTC.

__a. Locate your Share code with Jazz Source Control work item if it is not already open in the Work Items view.

__i. In the EEBAW JKE Banking project area, expand **Work Items > Shared Queries > Predefined** and double-click **Open assigned to me**. The Work Items view opens with your work item listed. If the project area was created with an ID other than your own, use the **Open Tasks** query instead and change ownership of the work item to your ID.

__b. Right-click your work item called `Share code with Jazz Source Control` and select **Set as Current Work Item**. All change sets will now automatically be associated with this task.

__c. Open your work item double-clicking it, set the Planned For to the current sprint (indicated with a → in the pull-down), and select from the status drop-down list the Start Working action.



__d. **Save** the work item.

__2. Import the mortgage application.

__a. In the Eclipse menu, select **File > Import.**

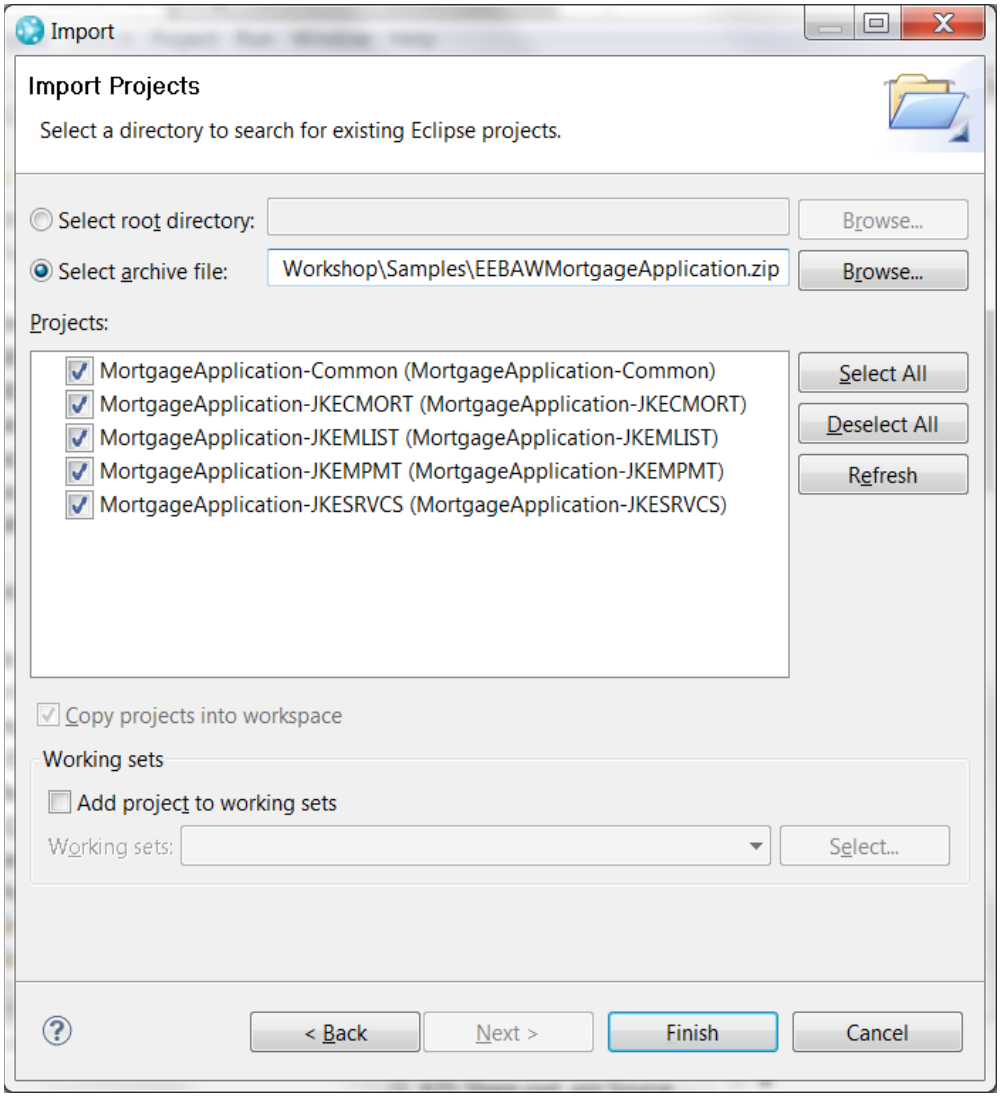
__b. In the Import wizard, expand General and select **Existing Projects into Workspace**. Click **Next**.

__c. Choose **Select archive file** and click **Browse**.

__d. The archive selection window opens:

__i. Navigate to the `EEBAWMortgageApplication.zip` file you downloaded as part of this workshop, select it, and click **Open.**

__ii. Select all of the projects in the archive. Click **Finish** to close the window and import the projects.



__3. Confirm the projects were successfully imported.

__a. Open the Resource perspective from the Eclipse menu by clicking **Window > Open Perspective > Other... > Resource**.

__b. Note the five new projects (in addition to the previously imported CommonBuildScripts project) in the Project Explorer view, decorated with the z icon to indicate that they are zComponent projects.

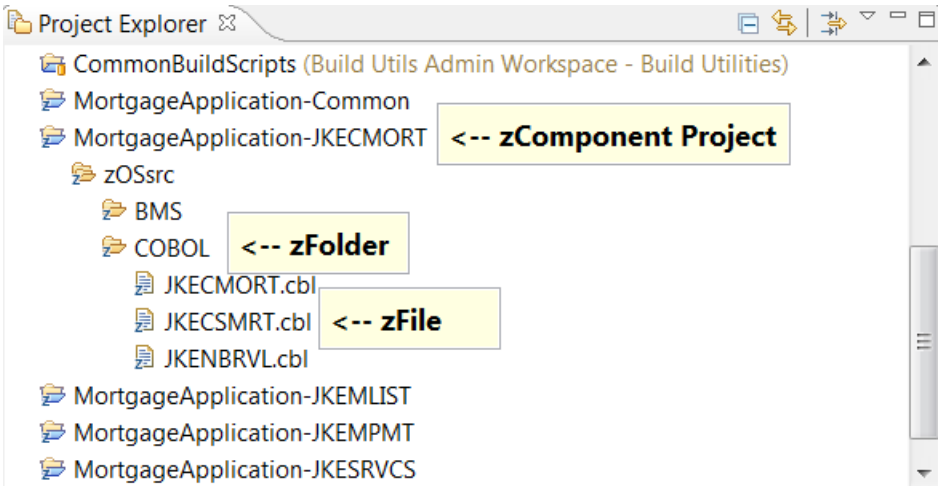
2.3 Explore the zComponent projects

Discover the structure of a zComponent project by exploring the projects you have imported into your eclipse workspace.

__1. Expand the project MortgageApplication-JKECMORT and note the zOSsrc folder. All files in zOSsrc subfolders represent MVS data set members.

__2. Expand the `COBOL` sub-folder. This COBOL sub-folder is referred to as a zFolder. All files in a zFolder should be of the same type, because they will all be loaded to the same MVS data set at build time.

__3. Double-click one of the files in the COBOL zFolder to open it in the Eclipse editor. These COBOL files are referred to as zFiles, and represent your MVS data set members.



2.4 Share the zComponent projects

Create a repository workspace that flows to the Production stream and share the zComponent projects.

__1. Create a repository workspace that flows to the Production stream.

__a. From the Resource perspective, open the Team Artifacts view by clicking **Window > Show View > Other**, and the **Team > Team Artifacts**.

__b. In the Team Artifacts view, in the EEBAW JKE Banking project area, expand the Source Control node and right-click your **EEBAW JKE Mortgage Production Stream**. Select **New > Repository Workspace.**

__c. Take the default Repository Workspace Name and click **Finish**.

__d. In the Load Repository Workspace dialog that pops up, leave the default of Find and load eclipse projects and click **Finish**. There are no eclipse projects in the workspace to load, but the workspace will now appear in the Pending Changes view.

Ummm... what did I just do?

Source code is organized and shared among team members using streams. A repository workspace is an individual member's sandbox for working on changes to be delivered to a stream. You've created a repository workspace to check your zComponent projects in to so you can then deliver them to the stream. Note that in Lab 1, you used a repository workspace that was generated during project initialization in order to share the setup scripts.

__2. Share the zComponent projects.

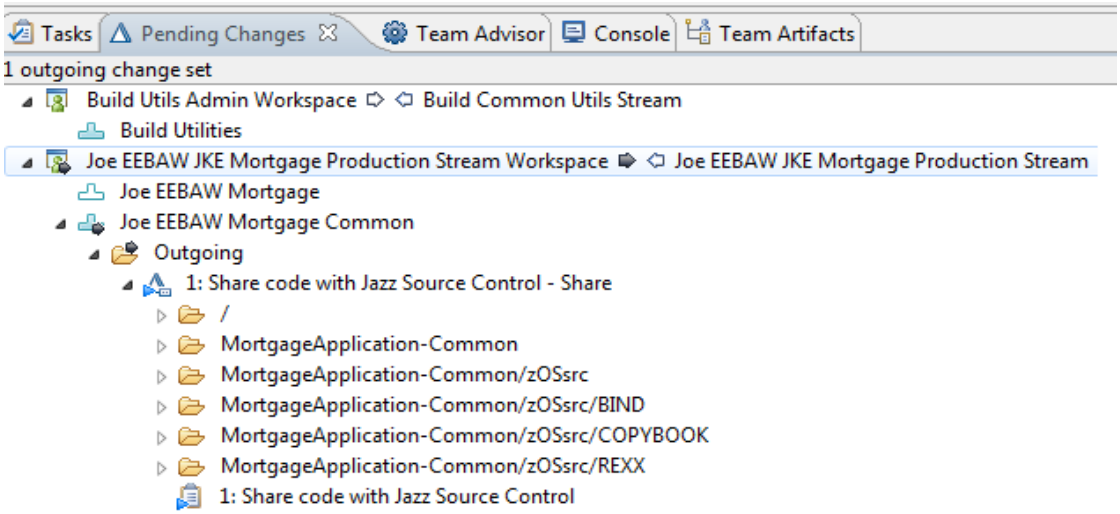
__a. In the Project Explorer view, right-click MortgageApplication-Common and choose **Team > Share Project...**. The Share Project wizard opens.

__b. Select J**azz Source Control** for the repository type and click **Next**.

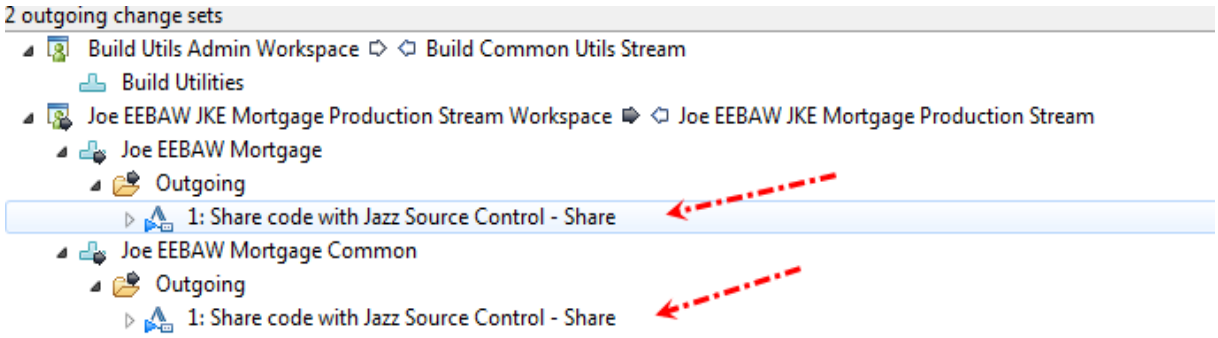
__c. On the Select Component page, ensure your repository is selected in the repository pull-down.

__d. Under Select a component or folder in an existing repository workspace, select the **EEBAW Mortgage Common** component in the workspace you created in step 1.

__e. Click **Finish** to complete the share wizard. You should now see an outgoing change set in the Pending Changes view.



__3. Repeat step 1 to share the remaining four “MortgageApplication-*” projects. Note that all four projects can be selected and shared at once.
Use the same workspace, but this time share to the **EEBAW Mortgage** component. Notice in the Pending Changes view that you now have two change sets, one for each component that you have shared projects to. You will deliver these changes to the stream in a future step.

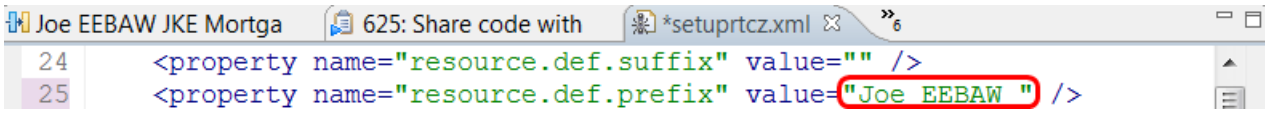


2.5 Associate the mortgage application with system definitions

You will now associate the zFolders in the mortgage application with “Destination data set for a zFolder” data set definitions. Recall that these data set definitions tell us how to create the data sets on the host that will hold the source that will be loaded from source control to be built. You will also associate the zFiles with language definitions, which specify how each file should be built. Language definitions will be covered in depth in the next lab.

__1. In the Project Explorer view, expand the MortgageApplication-Common project and double-click `setuprtcz.xml` to open it in the Ant editor. In the Ant editor:

__a. Look in the file for the property with name “resource.def.prefix”. You can use **Edit > Find/Replace** menu or **Ctrl+F** to search in the file. Ensure you set this property to the same value that you used in setupCommonSystemDefs.xml used in Lab 1.



__b. Continue to scroll through the contents of the Ant script, and see that in this case we are not creating any new system definitions but instead are associating existing system definitions with our zFolders and zFiles.

__c. Save your changes using **Ctrl-S** and close the editor.

__2. Run the setup script to associate system definitions with the mortgage application zComponent projects.

__a. Right-click the `setuprtcz.xml` file and select **Run As > Ant Build**... to open the Edit Configuration wizard:

__i. On the Targets tab, confirm that the “all” target is selected to execute.

__ii. On the JRE tab, confirm that Run in the same JRE as the workspace is selected.

__iii. Click **Run** to close the wizard and execute the Ant script.

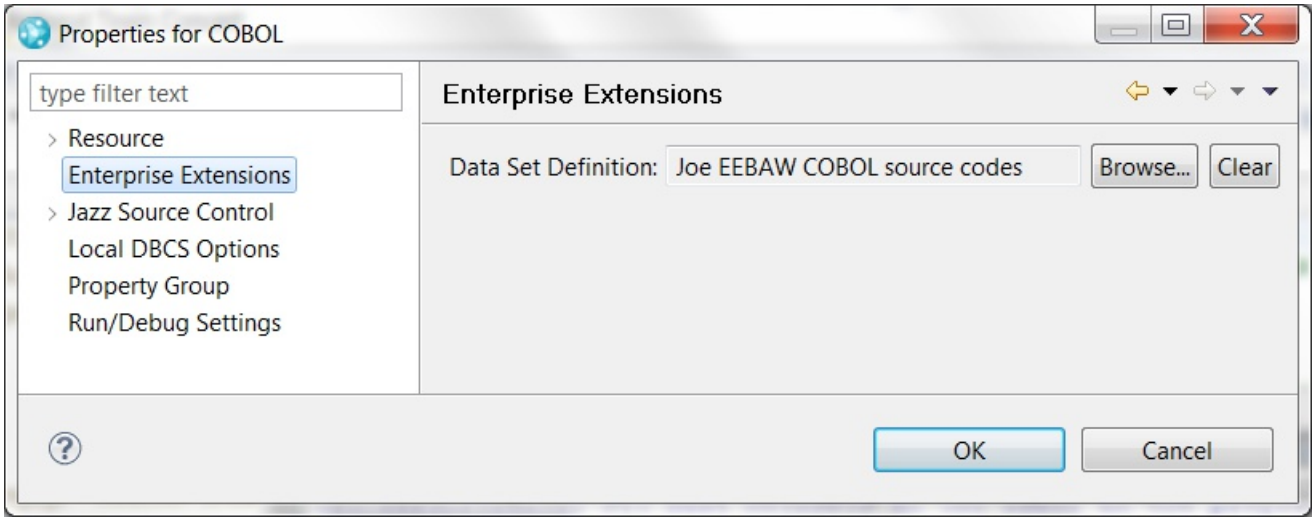
__b. Observe the execution of the system definitions generator in the Console view. The tool should complete with a message like the following:

```
all:
BUILD SUCCESSFUL
Total time: 7 seconds
```

__3. Verify that the system definitions were associated properly.

__a. In the Project Explorer view, expand MortgageApplication-JKECMORT > zOSsrc and right-click on the COBOL zFolder. Select **Properties**. The Properties page opens:

__b. In the left-hand navigator, click on Enterprise Extensions. The data set definition associated with this zFolder is displayed.



__c. Click **Cancel** to close the Properties page.

2.6 Deliver the mortgage application to the Production stream

You will check in the changes created by the system definitions generator and deliver the mortgage application to the EEBAW JKE Mortgage Production Stream.

__1. Open the Pending Changes view and notice that you now have some Unresolved changes in your workspace. Right-click on the Unresolved node under the EEBAW Mortgage component and select **Check-in and Deliver**. The Check-in and Deliver wizard opens:

__a. In the Change set comment window, enter `Adding mortgage application to source repository`. Click **Finish**.

__b. Repeat these steps for the EEBAW Mortgage Common component.

2.7 Summary

In this lab you have created the basic structure in Jazz SCM to support the development and build of mortgage applications. Based on the decisions made in Lab 1, you have created the components and the Production stream, and put the mortgage application source code under Jazz SCM control following this logical structure.

You have also associated the zFolders with data set definitions that will provide the information necessary to load the source code to partitioned data sets on the host. You have associated the zFiles with the language definitions that will specify how each file should be built, which you will learn about in the next lab.

Lab 3 Migrating your build to Rational Team Concert

In this lab, you will discover the tasks involved to migrate your existing build infrastructure to Rational Team Concert's dependency build capability.

In Lab 2, you learned about data set definitions for referencing all MVS data sets involved during the build process. Two other types of system definitions – translators and language definitions – will be reviewed in this lab. The dependency build leverages the information contained in translators and language definitions to perform the compiles, links, binds, and any other steps necessary to build your application.

The steps we will follow are:

- Review the translators created by the system definitions generator and create a new translator
- Review the language definitions created by the system definitions generator and create a new language definition
- Create a dependency build definition

3.1 Define build operations with translators

Translators represent operations that will be performed on a file during a build, such as a compile or link-edit. The translator captures such configuration information as the actual module to be used for the operation, all of your DD allocations and concatenations such as SYSLIB, default options, and a maximum return code.

You will now review the translators that were generated in the last lab, and then convert a link-edit JCL sample to a new translator.

3.1.1 Explore the generated translators

__1. Open the Rational Team Concert eclipse client if not already open, and switch to the Work Items perspective.

__2. In the Team Artifacts view, expand **EEBAW JKE Common Build Utilities > Enterprise Extensions > System Definitions > z/OS > Language Definitions > Translators**. Double-click **EEBAW COBOL compilation (CICS&DB2)**. The Translator editor opens:

   __a. In the Call Method section, note that this translator is calling the COBOL compiler (represented using a data set definition) with a set of default options and a maximum return code of 4.

   __b. In the Data Set Properties section, notice all of the DD allocations and concatenations configured. DD allocations connect data sets to a program (in this case, the COBOL compiler) for a particular use, such as providing input or directing output. DD concatenations allow a program to utilize a series of data sets for a particular use, such as a search path for source code or libraries.

   __c. Close the Translator editor.

__3. Explore any of the other generated translators.

3.1.2 Create a new translator

Next you will create a new translator from JCL. Many host-based build technologies utilize either JCL, JCL plus proprietary extensions, or ISPF skeletons that expand to JCL in order to perform a build. As you convert this JCL to a translator, consider how migrating your own existing build infrastructure might be a similar process.

The sample JCL below performs a link-edit using a link card.

```
//LINKJCL JOB accountNo,userId,CLASS=A,MSGCLASS=Y,NOTIFY=&SYSUID
//         SET PROGRAM='progName'                          PROGRAM NAME |
//         SET HLQ='hlq'                                                |
//         SET CICSPRFX='CICSTS.V3R2M0'                                 |
//*-------------------------------------------------------------------
//LKED   EXEC PGM=IEWL,REGION=1024K,PARM='MAP,RENT'
//SYSLIB   DD DSN=&HLQ..OBJ,DISP=SHR

//          DD DSN=SYS1.CSSLIB,DISP=SHR
//          DD DSN=CEE.SCEELKED,DISP=SHR
//          DD DSN=&CICSPRFX..CICS.SDFHLOAD,DISP=SHR
//SYSLIN   DD DSNAME=&HLQ..LINK(&PROGRAM),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DSNAME=&HLQ..LOAD(&PROGRAM),
//            SPACE=(CYL,(1,1)),UNIT=VIO,DISP=(MOD,PASS)
//SYSUT1   DD UNIT=VIO,SPACE=(TRK,(10,10))
```

__1. In the EEBAW JKE Common Build Utilities project area in the Team Artifacts view, expand **Enterprise Extensions > System Definitions > z/OS > Language Definitions** and right-click on **Translators**. Select **New Translator**. The Translator editor opens:

__a. In the Name field, type the name *your_user_id* `EEBAW Link-edit using linkage-editor deck`.

__b. Capture the LKED step.

__i. Under Call Method, click the **Browse...** button under Called Program to select the data set definition that represents the program used in this processor, IEWL.

__a. In the Select Data Set Definition window, expand the EEBAW JKE Common Build Utilities project area and double-click **EEBAW Link-editor**.

__ii. Locate the PARM parameters in the LKED step in the sample JCL above, and enter these parameters in the Default options field. Also add `SSI(@{ssi_info})` to the Default options to specify that the program object time stamp should be stored in the system status index to be used for dependency build and promotion.

__iii. Enter a Maximum return code value of `0`.

__c. Create the SYSLIB concatenation to include the compiled object decks, IBM callable service stubs, linkage editor library, and CICS runtime libraries.

__i. Under Data Set Properties, click the **Add...** button next to the DD concatenations table. The Add DD Concatenation dialog opens:

__a. Type `SYSLIB` for the name.

__b. Click **Add,** the Add Data Set Definition window opens.

__c. Select **Data Set definition** and click **Browse**. Select “EEBAW Object decks” from the EEBAW JKE Common Build Utilities project area. Then click **OK**.

__d. Repeat step b – c for EEBAW SYS1.CSSLIB, EEBAW CEE.SCEELKED, and EEBAW CICS.SDFHLOAD.

__e. Click **OK** to close the Add DD Concatenation dialog.

External data sets

To include your Production, QA, and or Test libraries in your SYSLIB, you would simply create “Existing data set used for build” data set definitions for these libraries and add them to the concatenation in the appropriate order.

__d. Create the SYSLIN allocation specify the location of the link-edit control card.

__i. Under Data Set Properties, click the **Add...** button next to the DD allocations table. The Add DD Allocation dialog opens:

__a. Type `SYSLIN` for the name.

__b. For the Data set, select **Translator input** to indicate that the link card data set member should be allocated as SYSLIN.

__c. Click **OK** to close the Add DD Allocation dialog.

__e. Create the SYSPRINT allocation for the link-edit listings.

RTC gives you the option to publish your build logs (all logs or failing logs) to the build result.

__i. Under Data Set Properties, click the **Add...** button next to the DD allocations table. The Add DD Allocation dialog opens:

__a. Type `SYSPRINT` for the name.

__b. Select **Data Set Definition** option and click **Browse...**Select “EEBAW Temporary file” from the EEBAW JKE Common Build Utilities project area. Click **OK**.

__c. Select the **Publish output** option.

__d. Click **OK** to close the Add DD Allocation dialog.

__f. Create the SYSLMOD allocation to define the data set to contain the output program objects.

__i. Under Data Set Properties, click the **Add...** button next to the DD allocations table. The Add DD Allocation dialog opens:

__a. Type `SYSLMOD` for the name.

__b. Select **Data Set Definition** option and click **Browse...** Select “EEBAW Program objects” from the EEBAW JKE Common Build Utilities project area. Click **OK**.

__c. Select the **Append member name to data set name** option.

__d. Select the **Save data set as output** option to indicate this is a build output.

__e. Click **OK** to close the Add DD Allocation dialog.

__g. Create the SYSUT1 allocation to define the work file (scratch dataset).

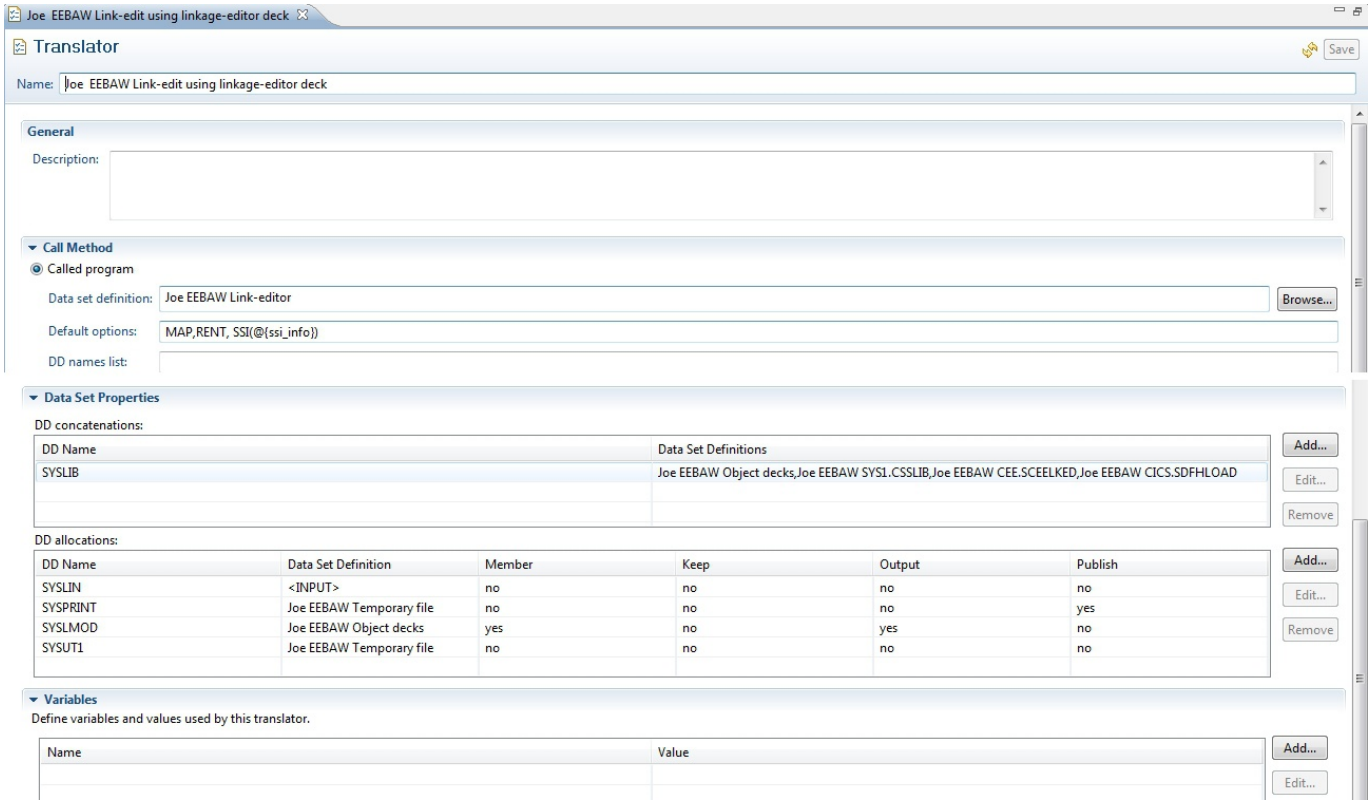
__i. Under Data Set Properties, click the **Add...** button next to the DD allocations table. The Add DD Allocation dialog opens:

__a. Type `SYSUT1` for the name.

__b. Select **Data Set Definition** option and click **Browse..** Select “EEBAW Temporary file” from the EEBAW JKE Common Build Utilities project area. Click **OK**.

__c. Click **OK** to close the Add DD Allocation dialog.

__h. Confirm your translator matches the image below and save your translator using **Ctrl-S**.



3.2 Order build operations with language definitions

Language definitions specify the steps, or translators, to be performed on each buildable file during a build.

You will now review the language definitions that were generated in the last lab, create a new language definition that utilizes the translator you just created, and then examine the various methods available for associating a file with a language definition.

3.2.1 Explore the generated language definitions

__1. In the Team Artifacts view, expand **EEBAW JKE Common Build Utilities > Enterprise Extensions > System Definitions > z/OS > Language Definitions**. Double-click **EEBAW COBOL compilation (CICS&DB2) and link-edit**. The Language Definition editor opens:

__a. On the General tab, in the Translators section, notice that this language definition specifies two translators, one to perform a COBOL compile and one to perform a link-edit. All files associated with this language definition will be compiled and link-edited using the options configured in the specified translators.

File-level overrides
You can override compile and link-edit options for individual files by using variables in your translators.

__b. On the Scanners tab, notice the Default Scanner for System z is specified. This is the scanner that is provided out of the box to scan your source code to gather logical dependency information to be used by the dependency build process.

Language definitions for non-buildable programs
Files that are not built but are included in the build of other programs, such as copybooks, need to be associated with a language definition to indicate that they need to be scanned for dependency information. Such language definitions would have no translators. See “EEBAW Copybook (no translators)” for an example of such a language definition.

__c. Close the Language Definition editor.

__2. Explore any of the other generated language definitions.

3.2.2 Create a new language definition

Next you will create a new language definition that references the translator you created above.

__1. In the EEBAW JKE Common Build Utilities project area in the Team Artifacts view, expand **Enterprise Extensions > System Definitions > z/OS** and right-click on **Language Definitions**. Select **New Language Definition**. The Language Definition editor opens:

__a. In the Name field, type *your_user_id* `EEBAW Link-edit`.

__b. For Language, select Link edit.

__c. In the Translators section, click **Add...** to open the Add Translator window:

__i. Select the **Translator** option and click **Browse**... to open the Select Translator window.

Using build properties for system definitions

Note the option to use a build property here instead of selecting a specific translator. This allows you to configure multiple build definitions that use the same language definitions but different translators, or to change your translator at build request time.

__ii. Select **EEBAW JKE Common Build Utilities > *your_user_id* EEBAW Link-edit using linkage-editor deck** and click **OK**.

Double check!

Ensure you selected the translator you just created above and not the EEBAW Link-edit translator that already existed. Otherwise you will have build failures later in the lab.

__iii. Click **OK** to close the Add Translator window.

__d. Switch to the Scanners tab.

__e. In the Source Code Scanners section, select Default Scanner for System z and click the **Remove** button. The out of the box scanner does not support link cards. Instead, a LINK dependency type is added here manually, and source code data created by the system definitions generator establishes the relationship between link cards and their included object decks. By doing this, we ensure that the link-edit occurs every time an object deck is rebuilt.

__f. In the Dependency Types section, remove the existing entries by selecting them all and clicking **Remove**.

__g. Add a new Dependency Type by clicking **Add...** in the Dependency Types section:

__i. Next to the Names pulldown, click **Create Type**...

__ii. In the Create Dependency Type window, type `Link`. Click **OK**.

__iii. Choose **Only the translators specified below** and select your EEBAW Link-edit using linkage-editor deck to associate this dependency type with your new translator.

__iv. Click **OK** to close the Add Dependency Type window.

__h. Save your new language definition using **Ctrl-S** and close the editor.

3.2.3 Assigning language definitions to files

Each file to be built or scanned must be associated with a language definition. A file can be associated with a language definition in a number of ways:

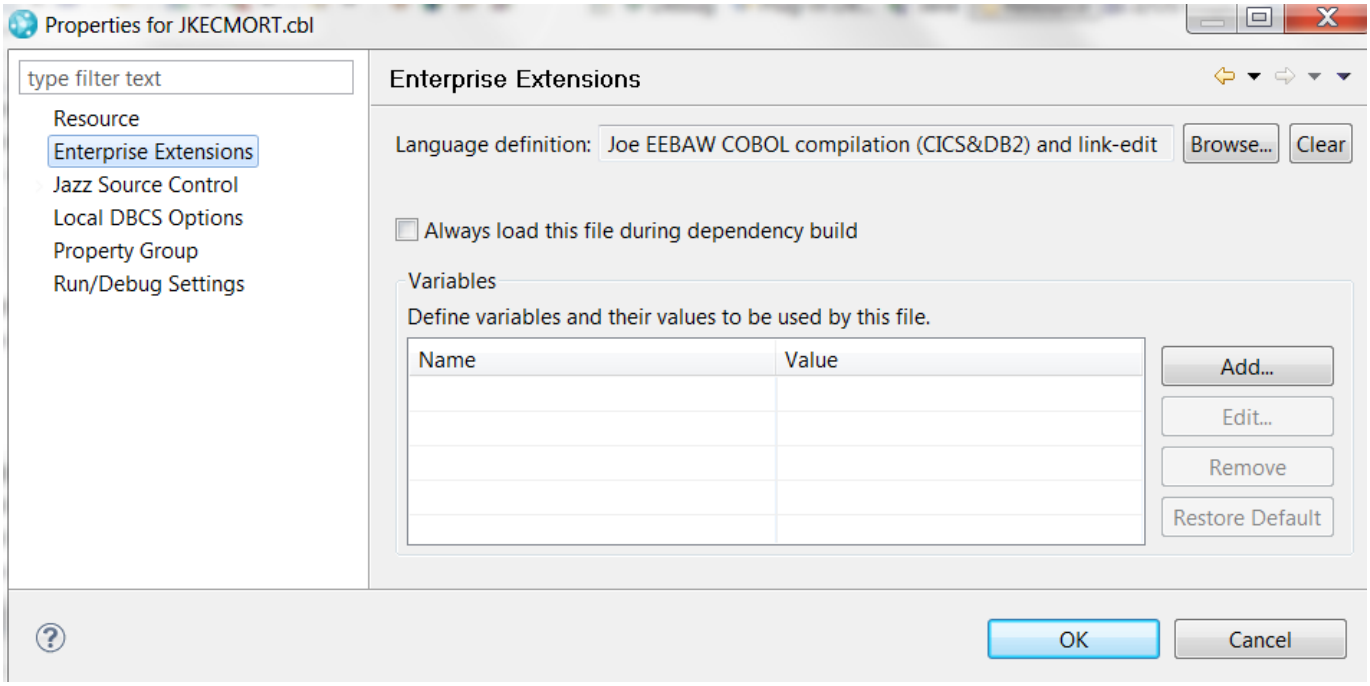
1. During zimport, language definitions can be optionally specified and assigned.
2. The system definitions generator can be utilized to associate language definitions to files, as you saw in the previous lab.
3. Each file can have its language definition viewed and modified in the file's properties on the Enterprise Extensions page.
4. A default extension can be specified in a language definition, and all files with that extension that do not have a language definition assigned will automatically be associated with the default language definition.
5. An Assign a Language Definition wizard is available to assign language definitions in bulk using rules.

You will review the associations created previously by the system definitions generator and associate the language definition you just created with a link card in the mortgage application.

__1. Review an association created by the system definitions generator.

__a. From the Resource Perspective, in the Project Explorer view, expand **MortgageApplication-JKECMORT > zOSsrc > COBOL** and right-click on `JKECMORT.cbl` file. Select **Properties**. The Properties page opens.

__b. In the left-hand navigator, click on Enterprise Extensions. The language definition associated with this zFile, EEBAW COBOL compilation (CICS&DB2) and link-edit, is displayed.



__c. Click **Cancel** to close the Properties page.

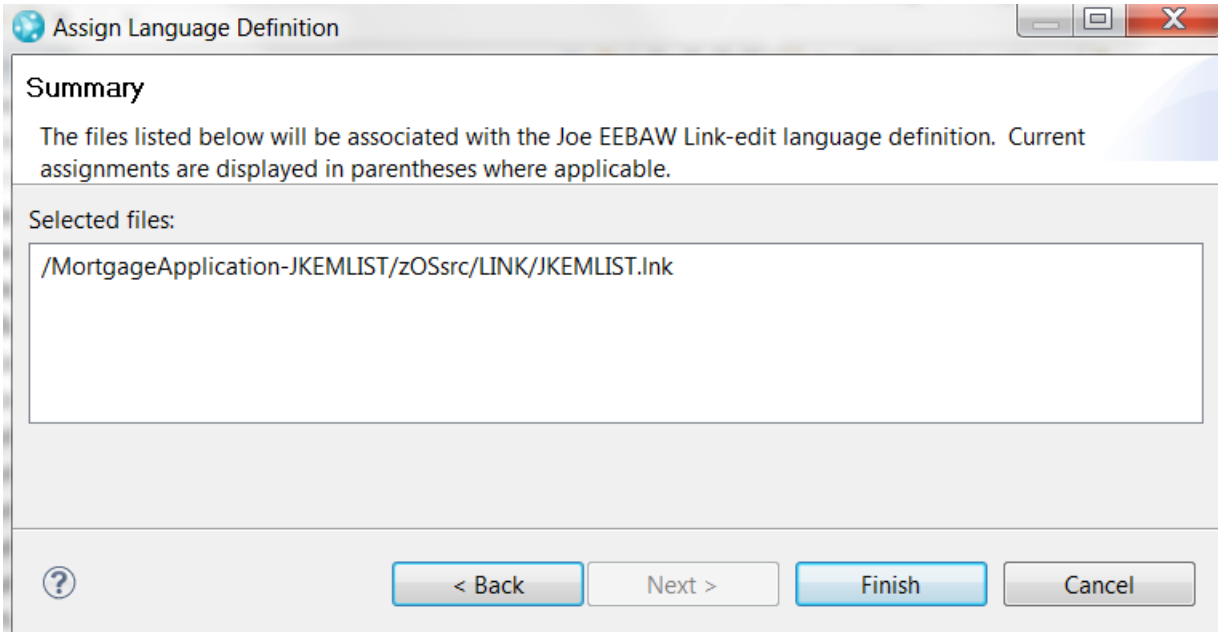
__2. Create a new association using the Assign a Language Definition wizard.

__a. In the Project Explorer view, select all five zComponent projects, right-click and select **Enterprise Extensions > Assign a Language Definition.**..

__b. On the first page of the wizard, click **Browse**... to choose a Language Definition. Select the language definition you just created called *your_user_id* `EEBAW Link-edit` and click **OK**.

__c. In Select files, click **Apply to all files that match**. In the Includes field, type `*.lnk`. Click **Next**.

__d. Confirm that there is one ".lnk" file that will be associated with your language definition. Click **Finish** to assign the language definition and close the wizard.



__3. You will now have an unresolved change in your Pending Changes view. Your file has a Modified property label, and that modified property can be viewed by double-clicking the file in the Pending Changes view to open the Compare editor.

__4. Check-in your change, associate the change set with your Share code with Jazz Source Control work item (if it is not automatically associated), and deliver to the Production stream.

3.3 Create the dependency build definition

You have successfully stored your source in the repository, created the system definitions necessary to build the source, and associated the source with the system definitions. Now you will create the dependency build definition to drive the build of the mortgage application in the Production stream.

__1. From the Work Items perspective, in the Team Artifacts view, expand the EEBAW JKE Banking project area, right-click **Builds**, and select **New Build Definition...** The New Build Definition wizard opens:

Is your head starting to spin?

We have been jumping back and forth between the Resource and Work Items perspective. You may find that you want to customize one perspective with all of your most commonly used views to avoid the context switching day to day.

__a. Click the **Browse...** button to select your EEBAW JKE Banking project area and click **OK**.

__b. Use the default option **Create a new build definition** and click **Next.**

__c. For the ID, type the name *your_user_id*.eebaw.mortgage.prod.

__d. In Available build templates, scroll and select z/OS Dependency Build – Rational Build Agent. Click **Next**.

__e. Click **Next** until the end of the wizard, noting the configuration options available with this build template. Click **Finish** to close the wizard.

Build definition ownership

Your build definition is owned by the project area. You can change the owner to a specific team to control who has permission to modify the definition and request builds.

__2. The Build Definition editor opens the new definition to be edited and saved:

__a. On the Overview tab in the Supporting Build Engines section, click **Create...** to open the New Build Engine wizard:

__i. For the ID, type the name *your_user_id*.eebaw.rba.engine.build.

__ii. Under Available build engine types, select **Rational Build Agent**.

__iii. Click **Finish** to create the build engine. We will edit the new build engine later.

__iv. If a build engine called “default” exists, highlight it and click **Remove**.

One build at a time!
Allowing multiple concurrent team builds of your dependency build may result in unnecessary rebuilds of the same program. You should prevent this by only configuring one supporting team build engine. Additional engines may be added for personal builds only.

__b. If you are running this workshop on a server with a host name that is not known to the machine where you will be running your build agent, go to the Properties tab to create a build property for the repository address.

__i. Next to the Properties table, click the **Add...** button.

__ii. On the Add Build Property panel, select type String and click **OK**.

__iii. For the Name, type `repositoryAddress`.

__iv. For the Value, type `${repo.url}`. We will provide a value for ${repo.url} in the build engine so it can be reused in other build definitions.

__v. Click **OK** to add the property.

__c. Review the Job Output Publishing options on the Output Publishing tab.

__d. On the Jazz Source Control tab, create a new repository workspace that flows with the Production stream. This workspace should be dedicated to this build definition and not used elsewhere.

__i. In the Build Workspace section, click **Create...**. The New Repository Workspace wizard opens:

__a. On the Select a Stream page, select **Flow with a stream**.

__b. Expand the EEBAW JKE Banking project area and select EEBAW JKE Mortgage Production Stream. Click **Next**.

__c. On the New Repository Workspace page, enter the name *your_user_id*`.eebaw.mortgage.prod Build Workspace`. Click **Next**.

__d. On the Read Access Permission page, choose Scoped and choose the EEBAW JKE Banking project area. This gives your Jazz build functional ID read access to the workspace. You could also make the workspace private and set the ownership to the Jazz build functional ID. Click **Next**.

__e. Leave both components selected and click **Finish** to create the workspace and close the wizard.

__e. In the Load Options, specify a USS directory and high level qualifier for loading your USS files and data sets.

__i. In Load directory field, type a location that is writeable by the TSO user used to execute the builds, such as `${build.directory}/`*`your_user_id`*`.eebaw.mortgage.prod`, where "build.directory" is a build property containing the common root for all of your build, promotion, and deployment definition USS directories. We will specify a value for build.directory in the build engine below. If you are performing this workshop in a classroom setting, your instructor will provide you with this value.

__ii. In the Resource prefix field, type a high level qualifier that is writeable by the TSO user used to execute the builds, such as `BUILDER.`*`YOUR_USER_ID`*`.PROD`. Each segment in the resource prefix must be 8 characters or less. If you are performing this workshop in a classroom setting, your instructor will provide you with this value.

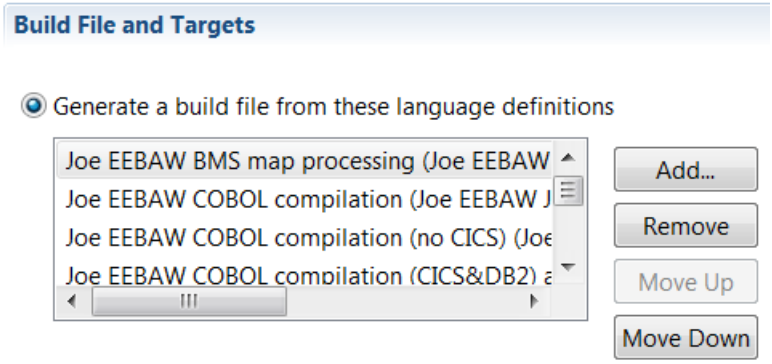
__f. On the z/OS Dependency Build tab, in the Build File and Targets section, specify the language definitions to be processed. All files associated with these language definitions will be built in the order specified here.

__i. Click the **Add...** button to open the Select Language Definition window.

__ii. Expand the EEBAW JKE Common Build Utilities project area and select all of the language definitions.

__iii. Click **OK** to close the window.

__iv. Ensure that all BMS maps are built first, followed by all COBOL subprogram compiles, and all link edits occur last (i.e. if it has the words link edit in the name, make sure it is after all of the language definitions that do NOT have link edit in the name). Use the **Move Up** and **Move Down** buttons as necessary to adjust the build order. Sorry we can't seem to get a screen capture that shows everything!



__g. Review the options on the Dependency Options tab. A build subset allows you to limit the selection of programs considered for build.

__h. Save the build definition via **Ctrl-S** and close the Build Definition editor.

3.4 Configure the Rational Build Agent build engine

The build engine is an artifact created in the Jazz repository to represent the actual Rational Build Agent running on the build machine. The engine contains the information necessary to connect to the agent when dependency builds (or other builds supported by the Rational Build Agent) are requested.

You will now configure the Rational Build Agent build engine that you created in the previous step.

__1. In the Team Artifacts view, expand **EEBAW JKE Banking >Builds > Build Engines.** Right-click the engine you created in the last step called *your_user_id*.eebaw.rba.engine.build, and select **Open Build Engine** to open the Build Engine editor:

__a. Create a build property for build.directory.

__i. On the Overview tab, in the Properties section, click **Add...**.

__ii. On the Add Build Property panel, select type String and click **OK**.

__iii. For the Name, specify `build.directory`.

__iv. For the Value, specify a USS location writeable by the TSO user used to execute the builds, such as `/u/build/eebaw/mortgage.`

__v. Click **OK** to add the property.

__b. If you utilized the repositoryAddress property in your dependency build definition, you must now set a value for repo.url.

__i. Repeat the steps above, using the following values:

__a. Name: `repo.url`

__b. Value: `https://`*`your_clmServer_ip_address`*`:9443/ccm`, where the IP address for your repository has to be reachable from your build machine. Update the port and context root as appropriate.

__c. On the Build Agent tab, configure the information required to connect to the Rational Build Agent running on your z/OS build machine. If you are following this workshop in a classroom environment, your instructor will provide the necessary values to you.

__i. Specify the host name and port where your Rational Build Agent is running.

__ii. Provide a user name and password to connect to the running agent. You can read in the Information Center about security considerations and the authority under which your builds will execute.

__iii. Test the connection by clicking **Test Connection** in the Build Agent Connection Test section. A successful connection will look like the following:

```
Message: Socket created successfully.
Authentication Pass.

Platform: os/390 23.00 03
Version: 7.1.3.0-0-0004
PingResult: PingOk
ExitStatus: 0
```

__d. Save the build agent using **Ctrl-S** and close the editor.

3.5 Verify the build configuration

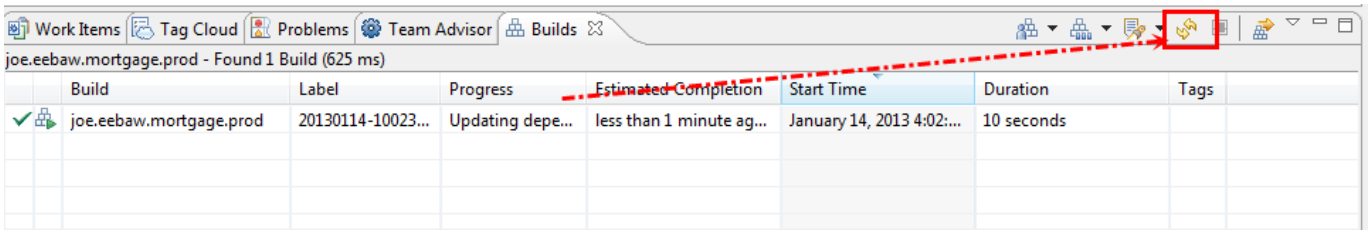
You will now verify that your build is properly configured by requesting a Production build. This will build all of the programs in the Production stream. The purpose of this exercise is not only to confirm that the build is properly configured, but also to seed the dependency build so that all future build requests will only build changed and impacted artifacts. Workarounds are available if it is not desirable or not possible to do a full build. See the accompanying workshop presentation for options.

3.5.1 Request a build

__1. In the Team Artifacts view, expand **EEBAW JKE Banking > Builds > Build Definitions**, right-click your new dependency build, and select **Request Build...**.

__2. In the Request Build wizard, click **Submit**.

__3. The Builds view opens to display the status of your build request. You can update this status by clicking the Refresh icon in the upper right-hand corner of the Builds view.



__4. The build result can be opened at any time (as long as it has started processing, i.e. it is out of “Pending...” state in the Progress column) by double-clicking the entry in the Builds view. The build is finished when the Progress column reads “Completed.”

3.5.2 Review the build result

__1. Double-click your build result to open the Build Result editor.

__2. Click on the Activities tab and notice the activities that occurred and the duration of each activity.

__3. On the Compilation tab, expand the entries in the Compiles table and select individual file names to view warnings produced by their compilation. You should see an entry for each COBOL program compiled during this build.

__4. On the Logs tab, you can find the `build-`*`timestamp`*`.log` file, which you may need to review to troubleshoot failed builds. If a translator fails, you will find that listing here. To publish successful listings on this tab, deselect the Publish only when an error occurred option on the Output Publishing tab of the build definition.

__5. The External Links tab contains a link to the build report. Click this link to review the build report.

Build report

The build report provides a summary of all files that were built and why. Clicking the output column of each entry will open a build map, which shows all of the inputs and outputs from the build of that file.

__6. Click back to the Overview tab and notice that the build report link is available here as well, and details how many files were processed during this build.

__7. Continue to explore the build result, and then close the build result editor.

How to know your build did what you expected

In summary, there are several sources of information in, and attached to, the build result to tell you if your build performed as expected.

1) Compilation tab – Shows you which programs were compiled, and what errors and warnings they produced
2) Logs tab – Listings for failed translator steps are published here. Successful listings for each step can also be published but by default they are not. Also, the main build log can be found here and contains logging of the entire build process. You can add a -debug flag to the Ant arguments field of your build definition for even more verbose output.
3) External links tab – The buildReport.xml tells you what was built, *why* it was built, and contains links to the build maps telling you the inputs and outputs for each program.

3.6 Summary

In this lab, you have reviewed the purpose of translators and language definitions and created an instance of each. You have created a dependency build definition and the necessary supporting artifacts, including the build engine and dedicated build repository workspace. Lastly, you have performed a build of the mortgage application code at the Production stream level.

Lab 4 Promoting your changes from Development to Production

In this lab, you will configure your project area to allow changes to be promoted from Development up through Quality Assurance to Production. You will also learn how to use promotion to prime new releases with your Production level code.

The Rational Team Concert promotion capability facilitates the flow of your source code and build outputs together through your code hierarchy, and allows you to minimize the amount of rebuilding that has to occur at each level. It can be performed at two different granularities: component and work item. A typical usage, which we will configure in this lab, is to utilize component promotion to prime your new releases from Production, and to work item promote change requests up from Development through to Production. Component promoting down your hierarchy allows you to populate your stream for your new release with the Production-level code as a starting point, and it allows you to prime your dependency build such that only your changed and impacted programs are built going forward. Work item promotion, on the other hand, allows you to propagate individual change sets and their built outputs up from Development to Production.

The steps we will follow are:

- Create the Development and QA streams and dependency builds
- Configure a “primer” promotion definition for seeding lower level streams and builds from the Production level
- Prime QA and Development using component promotion
- Configure promotions from Development to QA and from QA to Production

4.1 Create the Development and QA streams

You currently have a fully populated Production stream. Now, you need to create the Development and QA streams. You will leave them empty and populate them from the Production stream using component promotion.

__1. If it is not still open from Lab 3, start your Rational Team Concert client and open the Work Items perspective.

__2. Create the Development stream.

__a. Within the EEBAW JKE Banking project area, right-click the **Source Control** node and click **New > Stream.**..

__b. Enter the name *your_user_id* `EEBAW JKE Mortgage Development Stream.`

__c. **Save** and close the editor.

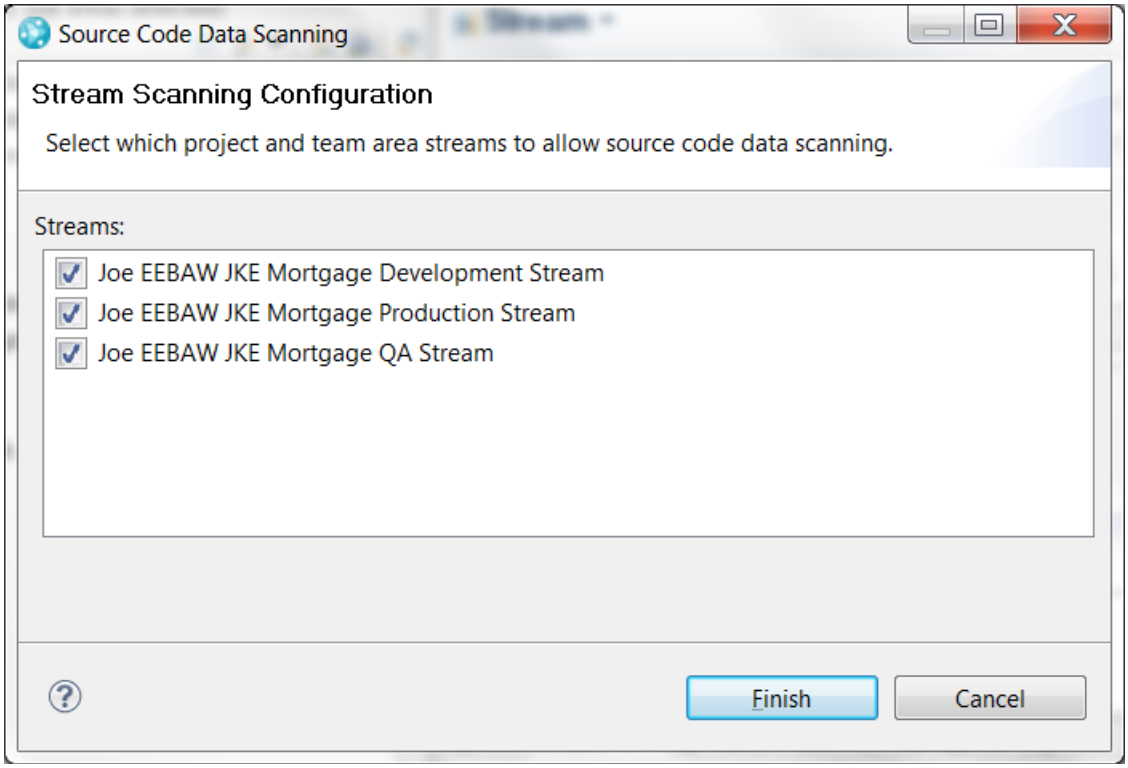
__3. Repeat step 2, this time entering the name *your_user_id* `EEBAW JKE Mortgage QA Stream.`

4.2 Turn on source code scanning for Development and QA streams

You learned in Lab 3 about the use of scanners to gather source code logical dependency information for the dependency build process. This source code scanning is enabled at the stream level. Scanning was turned on for the Production stream by the system definition generator script you executed. Now you will manually turn on scanning for your new Development and QA streams.

__1. In the Team Artifacts view, within the EEBAW JKE Banking project area, expand Enterprise Extensions, right-click **Source Code Data**, and select **Select Streams for Scanning...**

__2. In the Stream Scanning Configuration dialog, select your new Development and QA streams (and leave your Production stream selected as well). Click **Finish**.



4.3 Create the Development and QA dependency builds

You currently have a dependency build successfully configured at the Production level. Now, you need to create dependency builds at Development and QA.

But I'm not building at QA!

The Development-level dependency build will be used to build all of the new changes that developers will be delivering to the Development stream. The QA-level dependency build will be used simply to configure the promotions from Development to QA and QA to Production. You will not actually be building at the QA or Production level during normal development.

4.3.1 Create the Development dependency build

__1. In the Team Artifacts view, within the EEBAW JKE Banking project area, right-click Builds and select **New Build Definition...**

__2. In the New Build Definition wizard, select Create a build by copying and existing build definition and choose the ***your_user_id*.eebaw.mortgage.prod** definition you created in Lab 3. Click **Finish**.

__3. Configure the Overview tab.

__a. In the ID field, type *your_user_id*.eebaw.mortgage.dev.

__b. Under Supporting Build Engines, click **Add...** and select the build engine you configured in Lab 3 called *your_user_id*.eebaw.rba.engine.build.

__4. Configure the Jazz Source Control tab.

__a. Create a new, dedicated repository workspace that flows with the Development stream.

__i. In the Build Workspace section, click **Create...**. The New Repository Workspace wizard opens:

__a. On the Select a Stream page, select **Flow with a stream**.

__b. Expand the EEBAW JKE Banking project area and select EEBAW JKE Mortgage Development Stream. Click **Next**.

__c. On the New Repository Workspace page, enter the name *your_user_id*.eebaw.mortgage.dev Build Workspace. Click **Next**.

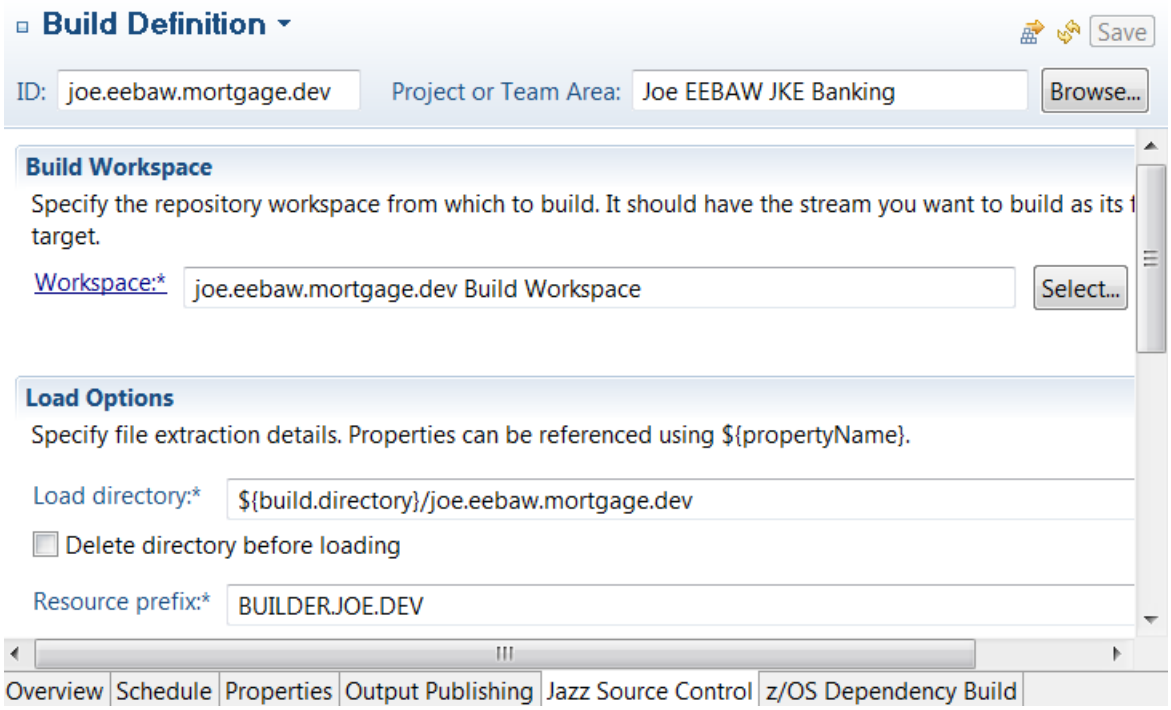
__d. On the Read Access Permission page, choose **Scoped** and choose the EEBAW JKE Banking project area. Click **Finish**.

__b. In the Load Options, specify a USS directory and high level qualifier for loading your USS files and data sets.

__i. In Load directory field, type a location that is writeable by the TSO user used to execute the builds, such as ${build.directory}/*your_user_id*.eebaw.mortgage.dev, where build.directory is the build property containing the common root that you created in Lab 3.
Ensure the value you choose is *different* from the value you used in your Production dependency build. If you are performing this workshop in a classroom setting, your instructor will provide you with this value.

__ii. In the Resource prefix field, type a high level qualifier that is writeable by the TSO user used to execute the builds, such as BUILDER.*YOUR_USER_ID*.DEV. Each segment in the resource prefix must be 8 characters or less.
Ensure the value you choose is *different* from the value you used in your Production dependency build. If you are performing this workshop in a classroom setting, your instructor will provide you with this value.

__5. **Save** the build definition and exit the editor. Take note that your build definitions differ only in their repository workspace and load destinations.



4.3.2 Create the QA dependency build

__1. Repeat the same steps as above, this time creating a workspace that flows to the EEBAW JKE Mortgage QA Stream, and replacing “dev” in all other values with “QA.”

4.4 Create the primer promotion definition

You will now create the promotion definition you will use to prime QA and Development from Production. Going forward, you will use the same technique to prime your new streams and dependency builds at the start of each release.

Primer promotion definition

Priming a new release is a one time action. That is, once you've primed your new release from Production by way of component promotion, you will not need to do it again. Therefore, rather than creating a new promotion definition every time you want to prime a new release, you can simply create one dedicated primer definition and update it for the appropriate source and target before each use. On the other hand, if your development strategy involves the use of component promotions for flowing changes through the system, you would in that case configure a dedicated promotion definition for each promotion level.

__1. In the Team Artifacts view, within the EEBAW JKE Banking project area, expand Enterprise Extensions, right-click **Promotions** and select **New Promotion Definition...** The New Promotion Definition wizard opens:

   __a. Ensure your EEBAW JKE Banking project area is selected.

   __b. Use the default option **Create a new promotion definition** and click **Next.**

   __c. For the ID, type the name *your_user_id*.eebaw.mortgage.primer.

   __d. Under Promotion Platforms, select **z/OS**.

   __e. Click **Finish** to close the wizard

__2. Configure the Overview tab.

   __a. Under Supporting Build Engines, click **Add...** and select the build engine you configured in Lab 3 called *your_user_id*.eebaw.rba.engine.build.

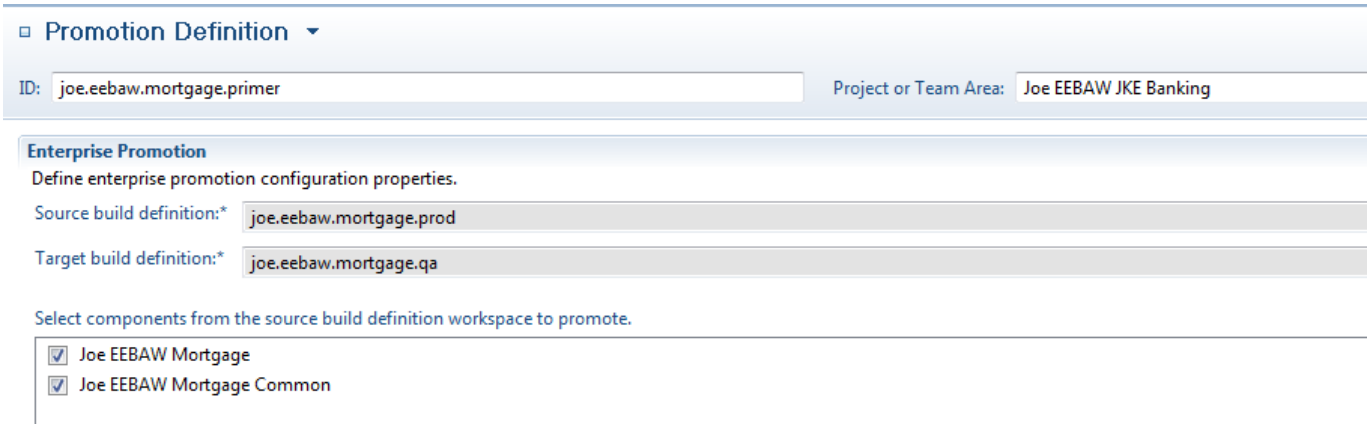
__3. Configure the Properties tab.

   __a. If you used the repositoryAddress property in your dependency build definition in Lab 3, configure it here as well.

__4. Configure the z/OS Promotion tab.

__a. In the Enterprise Promotion section, use the **Browse...** buttons to specify your Production dependency build as your Source and your QA dependency build as your Target.

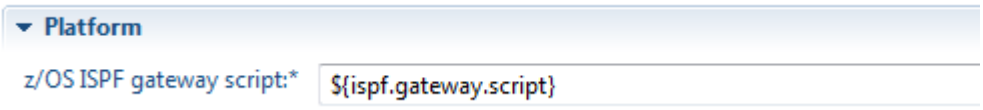
__b. Use the **Select All** button to select both components (Mortgage and Mortgage Common) to be component promoted.



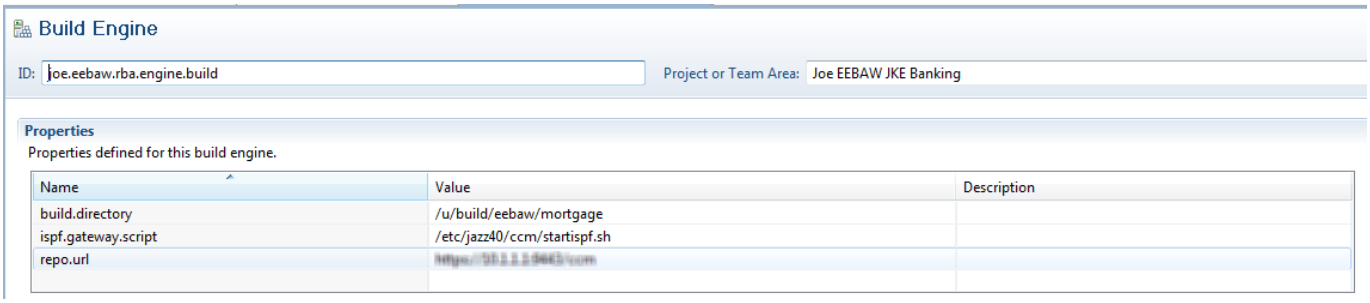
__c. In the Build Files and Targets section, specify a Destination directory such as `${build.directory}/`*`your_user_id`*`.eebaw.mortgage.primer`.
Any files generated by the promotion will be transferred to this USS directory on the build machine.

__d. Update the z/OS ISPF gateway script field with the location of your ISPF gateway startup script. The ISPF gateway is used to execute the REXX exec that actually copies your build outputs from your source data sets to your target data sets. Since you will be needing to set this value in other promotion definitions, as well as package and deployment definitions in the next lab, it is best to use a build property.

__i. In the z/OS ISPF gateway script, type `${ispf.gateway.script}`.



__ii. Add the ispf.gateway.script build property to your build engine, following the same steps laid out in Lab 3 for adding build.directory. Ask your instructor or system administrator for the appropriate value, such as `/etc/jazz40/ccm/startispf.sh`.



__5. **Save** the promotion definition using **Ctrl-S** and close the editor.

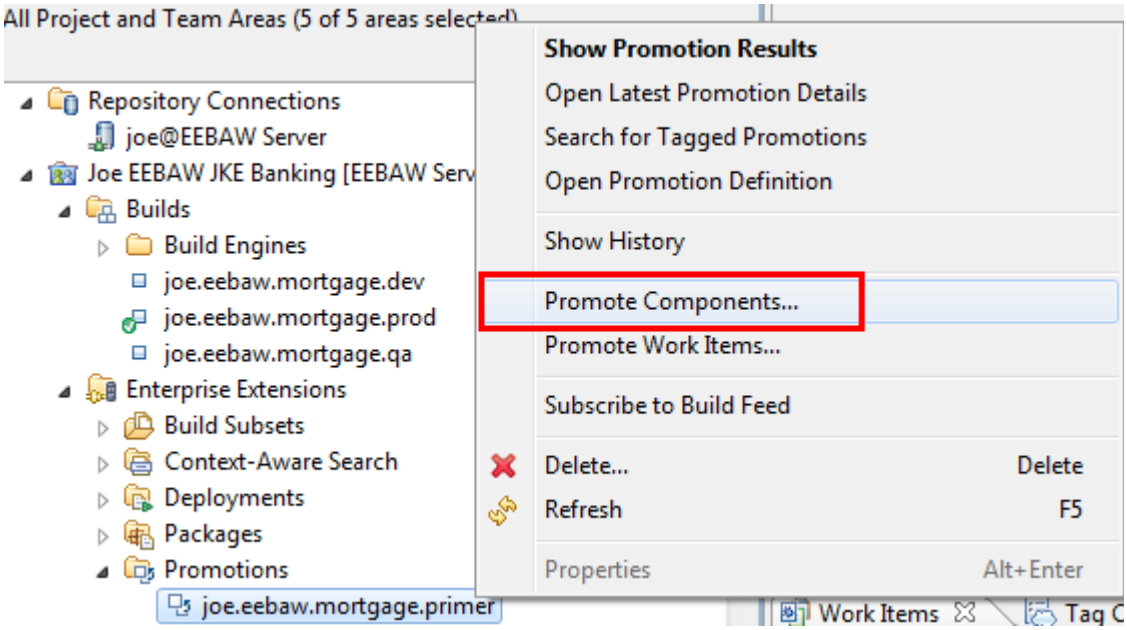
4.5 Prime QA and Development from Production

You will now perform component promotions to prime QA and Development from Production.

4.5.1 Component promote from Production to QA

__1. Request a component promotion from Production to QA.

__a. In the Team Artifacts view, expand **EEBAW JKE Banking > Enterprise Extensions > Promotions**, right-click your primer promotion definition, and select **Promote Components...**.



__2. In the Request Build wizard, confirm both components are selected and click **Submit**.

__3. The Builds view opens to display the status of your promotion request. You can update this status by clicking the Refresh icon in the upper right-hand corner of the Builds view.

__4. The promotion result can be opened at any time (as long as it has started processing, i.e. it is out of “Pending...” state in the Progress column) by double-clicking the entry in the Builds view. The promotion is finished when the Progress column reads “Completed.”

“Promotion” vs “Build”

Promotion leverages the RTC build infrastructure, and therefore you will sometimes see the word “Build” instead of “Promotion” in the views and wizards you use to perform your promotion.

4.5.2 Review the promotion result

__1. Open the promotion result and explore the contents.

__a. Double-click your promotion result in the Builds view to open the Build Result editor.

__b. On the External Links tab, you will find the links for the build maps of all of the promoted buildable files. You can click each link to see the build outputs that were copied to your QA data sets.

__c. On the Logs tab, you can find the `buildComplete-`*`timestamp`*`.log` file, which logs the promotion of the source code and source code data to QA.

__d. On the Downloads tab, you can find the `promotionInfo.xml` file. This is the file that is passed to the build machine, containing all of the outputs that are to be copied and their source and target data sets.

__e. Continue to explore the promotion result, and then close the build result editor.

__2. View the contents of the QA stream.

__a. In the Team Artifacts view, in the EEBAW JKE Banking project area, expand Source Control and right-click the **EEBAW JKE Mortgage QA Stream**. Select **Show > Repository Files**.

__b. In the Repository Files view that opens, navigate the Mortgage and Mortgage Common components and confirm that the QA stream is now populated. If the Repository Files view is empty, right-click the stream in the Team Artifacts view and click **Refresh**.

Bug alert!

Don't be alarmed if you notice that your QA build definition is showing an error status. The component promotion triggers a build at the target level by default (this can be suppressed with a build property). This build is failing as discussed in Promote Component is successful, but target dependency build definition shows failure (231295) . Fear not. Subsequent builds should succeed.

4.5.3 Component promote from Production to Development

__1. Update the primer promotion definition.

__a. Open the primer promotion definition (right-click and **Open Promotion Definition**).

__b. On the z/OS Promotion tab, change the Target build definition to *your_user_id*.eebaw.mortgage.dev.

__c. Make sure the components are checked.

__d. **Save** and close the promotion definition.

__2. Follow the steps above to request the component promotion from Production to Development and explore the promotion result.

Your QA and Development environments are now in sync with your Production environment with regards to the mortgage application.

4.6 Create the change request (up) promotions

You will now create the promotion definitions that will be used to flow changes from Development to QA to Production.

Component and work-item promotion definitions

The primer promotion definition is intended for priming lower level streams and build definitions through component promotion. The change request promotion definitions are intended to flow changes from Development all the way to Production through work-item promotion. However, note that you can request a component or work-item promotion from any promotion definition. The definitions themselves are not designated as one type of promotion or the other.

__1. Create the promotion from Development to QA.

__a. Create a copy of the primer promotion definition.

__i. In the Team Artifacts view, within the EEBAW JKE Banking project area, expand Enterprise Extensions, right-click **Promotions** and select **New Promotion Definition...** The New Promotion Definition wizard opens:

__ii. Ensure your EEBAW JKE Banking project area is selected.

__iii. Choose **Create a promotion definition by copying an existing definition**.

__iv. Select *your_user_id*.eebaw.mortgage.primer and click **Finish.**

__b. Configure the Overview tab.

__i. For the ID, type the name *your_user_id*.eebaw.mortgage.promote.devtoqa.

__ii. Under Supporting Build Engines, click **Add...** and select the build engine you configured in Lab 3.

__c. Configure the z/OS Promotion tab.

__i. In the Enterprise Promotion section, use the **Browse...** buttons to specify your Development dependency build as your Source and your QA dependency build as your Target. Note that you will have to choose the Target first since the editor will stop you from having the same build definition specified for Source and Target.

__ii. Select the Mortgage and Mortgage Common components. Note however that this setting is disregarded when performing a work item promotion.

__iii. In the Build Files and Targets section, specify a Destination directory such as `${build.directory}/your_user_id.eebaw.mortgage.devtoqa.`

__d. Save the promotion definition using **Ctrl-S** and close the editor.

__2. Repeat step 1 to create the promotion from QA to Production using the following values:

Field	Entry
ID	*your_user_id*.eebaw.mortgage.promote.qatoprod
Source build definition	*your_user_id*.eebaw.mortgage.qa
Target build definition	*your_user_id*.eebaw.mortgage.prod
Destination directory	${build.directory}/*your_user_id.eebaw.*mortgage.qatoprod

You will verify these definitions in Lab 6 when you test the end to end solution.

4.7 Summary

In this lab you have explored how the promotion feature works. You have created the promotion definition for priming new releases, and performed a component promotion to prime Development and QA from Production. You have also prepared the promotion definitions for moving future code changes from Development up the hierarchy to Production.

Lab 5 Packaging and deploying your application

In this lab you will perform the necessary configuration to be able to package your build artifacts and deploy them to your runtime environment using Rational Team Concert.

Prior to this lab, you configured the JKE Banking project area so the pilot application team can build the code using RTC Dependency Build, and promote the results through the defined hierarchy. Now you want to be able to deploy the build result artifacts into your QA and Production runtime environments. This implies gathering these resulting elements and packaging them, transferring or copying this package to the final runtime location, and performing the deployment activities.

The Packaging and Deployment features of Rational Team Concert, follow the same building blocks structure as the Dependency Build and the Promotion you have used in this workshop: you first create a package or deployment definition, then based on this definition you will be able to create a package or execute a deployment.

In this lab you will:

- Create a definition to package your build outputs at QA level
- Perform the configuration steps necessary to deploy the package contents in the QA runtime environment
- Review additional options and how these concepts would be extended to deploy in Production environment

Lessons learned:

- How the package and deployment features work in RTC
- How you can use these features to perform custom tasks needed in your deployment process

5.1 Create the package definition for QA environment

You have already seeded your QA environment with the contents of your application built at the Production stream level, including source code versions in the stream, build outputs in the QA data sets, and the associated metadata. You also have prepared the environment to promote changes from Development to this QA environment. Now you want to prepare your server to be able to package the built application modules.

Packaging Options

Rational Team Concert Enterprise Extensions offers two options for packaging your application: ship list based or work item based.

- Ship list packaging: allows you to define a list of data sets and/or members or a pattern to define the specific contents of the package
- Work item packaging: allows you to select the work items to consider for packaging. The package contents will be computed based on the outputs that the code change sets associated to the chosen work items generated. These outputs can be supplemented with a ship list.

__3. If it is not still open from the previous lab, start your Rational Team Concert client and open the Work Items perspective.

__4. Create the QA level package definition.

__a. Within the EEBAW JKE Banking project area, under Enterprise Extensions, right-click the **Packages** node and click **New > Package Definition...**

__b. Make sure the EEBAW JKE Banking project area is selected and click **Next**

__c. For the ID, type the name *your_user_id*`.eebaw.mortgage.package.qa`. Select **z/OS** as the Package File System and select **Finish**.

__5. Configure the package definition details opened after creation:

__a. Configure the Overview tab:

__i. Under Supporting Build Engines, click **Add...** and select the build engine you configured in Lab 3

__b. Switch to the Properties tab to configure it:

__i. If you used the repositoryAddress property in your dependency build and promotion definitions, configure it here as well.

__c. Go to the z/OS packaging tab:

__i. In the Package tab, click **Select...** for choosing a Build Definition. In the wizard that opens, highlight the build definition you created for QA level environment: *your_user_id*.eebaw.mortgage.qa. Then select **OK**.

__ii. The package root directory determines the USS directory where packages will be placed on your build machine. Following the used convention, enter the following value:
`${build.directory}/your_user_id.eebaw.mortgage.package.qa`

__iii. In the z/OS ISPF gateway script, type `${ispf.gateway.script}`

__iv. Create a default ship list definition:

Ship List in your Package Definition

You can define a default ship list for packaging in this wizard by adding filters and/or specific members in the Include/Exclude windows.

This default configuration can be overridden when you request the creation of the package.

You are going to define a default ship list and check how it works in the next lab.

You can find further information in this InfoCenter entry.

__a. Next to the Include window, click the **Add..** button. The Ship List Manager wizard opens.

__b. Click the **Add generic filter** option

__c. In the Data Set Filter field type the value for the load library using the resource prefix you configured previously for the QA level build:
`BUILDER.YOUR_USER_ID.QA.LOAD`

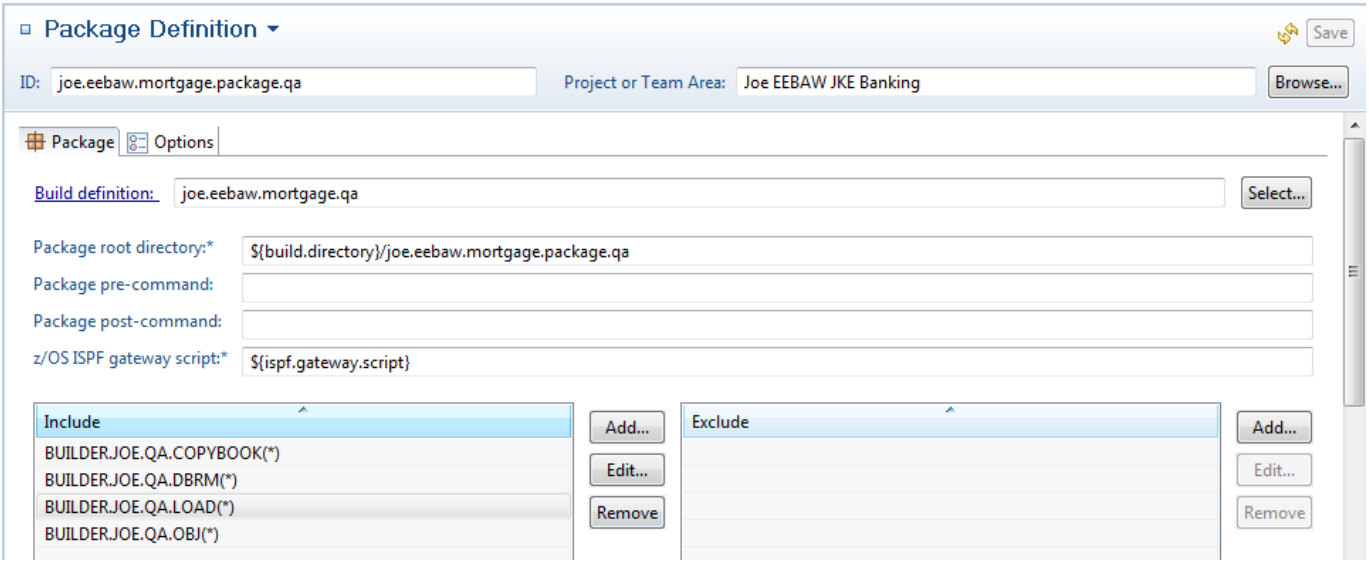
__d. In the Member Filter field type *, so all members of the PDS will be included.

__e. Click **OK**.

__f. Repeat steps a – e for the following Include values:

Data set filter	Member Filter
BUILDER.*YOUR_USER_ID*.QA.OBJ	*
BUILDER.*YOUR_USER_ID*.QA.COPYBOOK	*
BUILDER.*YOUR_USER_ID*.QA.DBRM	*

The result will look like the following:



Ship List Manager options

You have two options for defining the filters that will determine the members to be added:

- "Add Generic Filter": allows you to specify name pattern for PDS and members, allowing you to also use wild cards for the members selection.
- "Select Specific Members": allows you to run a query and select specific members from the returned results. You can use naming patterns and wild cards for the query.

  The wizard queries the build maps associated with the dependency build definition you selected in your package definition to find all outputs matching the name or pattern your specify. In this case, for example, any output that you built or promoted to the QA level that matches your specified pattern would be returned.

__d. Within the z/OS Packaging tab, change to the Options sub-menu tab:

__i. If entries were automatically added to the From PDS column, select and remove them. You will specify the restore mappings in the deployment definition.

Restore Mapping Table

By default, packaged contents are restored to the same data sets you saved them from when you ran the package. You can use this mapping table to deploy the packaged binaries to different data sets.

The restore mapping can be specified in several different places:

1. You can specify it in the package definition wizard. It will be used by default for all deployments using packages based on this definition.
2. When you request a package creation, the values can be defined or overridden.
3. This can also be specified in the deployment definition.

__ii. Leave Auto clean unchecked.

__iii. Optionally select any option to publish the packaging generated files with the packaging results.

Publish options

You can choose to have the packaging generated files attached to the package build result for direct download:

- “Publish manifest”: will attach the package manifest file to the package build result with the information of the resources packaged
- “Publish package”: will attach the zip package file itself to the package build result

__e. **Save** your changes and close the Package Definition editor.

You have configured a package definition for your QA environment. You will use this definition to create a package in Lab 6 when you test the end to end solution.

5.2 Create the deployment definition for QA environment

The next step is to prepare your project to be able to deploy your built application modules to your runtime environment. In this case you want to prepare the deployment of the mortgage application to your QA environment.

__1. Create the QA level deployment definition:

__a. Within the EEBAW JKE Banking project area, under Enterprise Extensions, right-click the **Deployments** node and click **New Deployment Definition...**

__b. Make sure the EEBAW JKE Banking project area is selected and click **Next**.

__c. For the ID, type the name *your_user_id*.eebaw.mortgage.deployment.qa. Select **z/OS** as the Deploy File System and select **Finish**.

__2. Configure the deployment definition details in the editor that opened after creation:

__a. Configure the Overview tab:

__i. Under Supporting Build Engines, click **Add...** and select the build engine you configured in Lab 3.

Deployment build engines

We are reusing our same build engine because we are performing the entire workshop on a single mainframe. If you were deploying to a different machine, you would need a build agent running there to process the deployment request and a corresponding build engine configured in your project area.

__b. Switch to the Properties tab to configure it:

__i. If you used the repositoryAddress property in your package definition, configure it here as well.

__c. Go to the z/OS Deployment tab:

Deployment definition phases

The deployment definition is based on configuring the three phases of the process: load the package to the server, deploy it and publish results.

The three sub-sections you find in this z/OS Deployment tab will provide you configuration options to tailor these three phases.

__i. In the Load sub-tab, click **Select...** for choosing a Package Definition. In the wizard that opens, highlight the package definition you created for QA level in the previous section: *your_user_id*.eebaw.mortgage.package.qa. Then select **OK**.

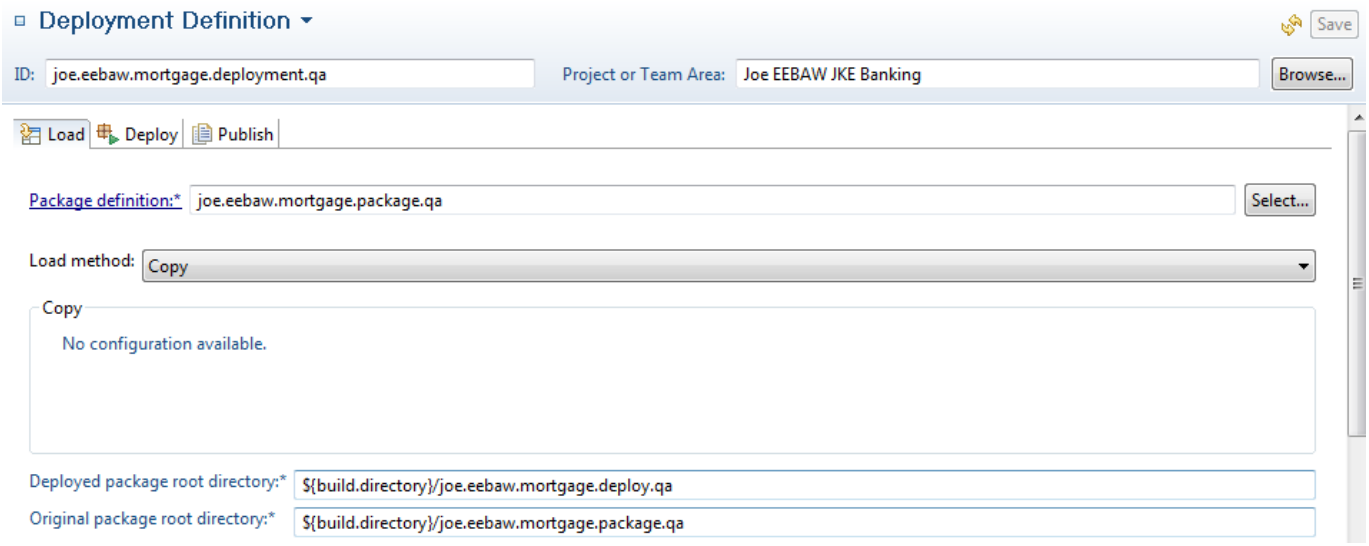
__ii. For Load method, change the default and select **Copy**.

Load method

This option specifies how you want to transfer the packages created to the destination environment: whether a network transfer via FTP or just a copy to a destination folder.

If you need to FTP it, you will have to specify the source server and credentials for the transfer.

__iii. For the Deployed package root directory, type ${build.directory}/*your_user_id*.eebaw.mortgage.deploy.qa.



__d. Within the z/OS Deployment tab switch to the Deploy sub-tab:

__i. Define the following restore mappings for the deployment. Click the **Add...** button and create one entry for each of the MVS data sets that hold build outputs. Be sure to update these example entries to match the resource prefix you defined for the build at QA.

From PDS	To PDS
BUILDER.*YOUR_USER_ID*.QA.OBJ	BUILDER.*YOUR_USER_ID*.QA.DEPLOY.OBJ
BUILDER.*YOUR_USER_ID*.QA.COPYBOOK	BUILDER.*YOUR_USER_ID*.QA.DEPLOY.COPYBOOK
BUILDER.*YOUR_USER_ID*.QA.LOAD	BUILDER.*YOUR_USER_ID*.QA.DEPLOY.LOAD
BUILDER.*YOUR_USER_ID*.QA.DBRM	BUILDER.*YOUR_USER_ID*.QA.DEPLOY.DBRM

Why is a restore mapping needed?

We are deploying on the same machine we used to build at Development and promote to QA. Without a restore mapping, our deploy would simply be overwriting the data sets that we populated when we promoted to QA. In a real scenario, you would be mapping to an actual runtime environment.

__ii. Note in this wizard you can define commands to be executed as part of the deploy and rollback process (Pre/Post Deploy and Rollback options). For example, you may want to trigger a DB2 BIND or CICS NEWCOPY after you deploy your DBRMs and load modules.

__e. Optionally specify any publish options available in Publish sub-tab within the z/OS Deployment tab:

Deployment Definition Publish options

You have some publish options available to attach manifest reports to the deployment build result. These are:

- “Publish delta deploy manifest” and “Publish cumulative deploy manifest”: will attach details on delta or cumulative deployed resources.
- “Publish rollback manifest”: in a deployment rollback execution will attach a manifest file containing the list of objects that were rolled back.

__f. **Save** your changes and close the Deployment Definition editor.

5.3 Optional: Configure packaging and deployment for Production

You want to also prepare the solution to deploy your application to the Production environment following what you already did for QA environment.

__1. Create the Production level package definition.

__a. Within the EEBAW JKE Banking project area, under Enterprise Extensions, right-click the **Packages** node and click **New Package Definition...**

__b. Verify that the EEBAW JKE Banking project area is selected

__c. Select **Create a package by copying an existing package** and highlight the QA level package definition called *your_user_id*.eebaw.mortgage.package.qa.

__d. Click **Finish**. The package definition opens.

__2. Modify the following values for the Production package definition:

__a. Configure the Overview tab:

__i. Modify the ID and name it following the convention: *your_user_id*.eebaw.mortgage.package.prod.

__ii. Under Supporting Build Engines, click **Add...** and select the build engine you configured in Lab 3 called *your_user_id*.eebaw.rba.engine.build.

__b. Switch to the Properties tab to configure it:

__i. Verify that the property called repositoryAddress is already there, if necessary.

__c. Go to the z/OS packaging tab:

__i. In the Package tab, click **Select...** for choosing a Build Definition. In the wizard that opens, highlight the build definition for the Production level environment: *your_user_id*.eebaw.mortgage.prod. Then select **OK**.

__ii. Define a package USS root directory for production packages following the convention: ${build.directory}/*your_user_id*.eebaw.mortgage.package.prod

__iii. Verify that the z/OS ISPF gateway script field contains the value: ${ispf.gateway.script}

__iv. Modify the copied QA default ship list: double click each entry and replace the “QA” qualifier for the data set filter with “PROD”. The values should be:

Data set filter	Member Filter
BUILDER.*YOUR_USER_ID*.PROD.LOAD	*
BUILDER.*YOUR_USER_ID*.PROD.OBJ	*
BUILDER.*YOUR_USER_ID*.PROD.COPYBOOK	*
BUILDER.*YOUR_USER_ID*.PROD.DBRM	*

__d. Within the z/OS Packaging tab, change to the Options sub-menu tab and perform the following changes:

__i. Restore Mapping table: if there are any entries in the restore mapping table, highlight them and click **Remove** to clear the table.

__ii. Leave Auto clean unchecked.

__iii. Optionally select any options to publish the packaging generated files with the package results.

__e. **Save** your changes and close the Package Definition editor.

__3. Create the Production level deployment definition:

__a. Within the EEBAW JKE Banking project area, under Enterprise Extensions, right-click the **Deployments** node and click New Deployment Definition...

__b. Make sure the EEBAW JKE Banking project area is selected.

__c. Select **Create a deployment by copying an existing deployment** and highlight the QA level deployment definition called *your_user_id*.eebaw.mortgage.deploy.qa.

__d. Click **Finish**. The deployment definition opens.

__4. Modify the following values for the Production deployment definition:

__a. Configure the Overview tab:

__i. Modify the ID and name it following the convention: *your_user_id*.eebaw.mortgage.package.prod.

__ii. Under Supporting Build Engines, click **Add..**. and select the build engine you configured in Lab 3 called *your_user_id*.eebaw.rba.engine.build.

__b. Switch to the Properties tab to configure it:

__i. Verify that the property called repositoryAddress is already there, if necessary.

__c. Go to the z/OS Deployment tab:

__i. In the Load sub-tab, click **Select...** for choosing a Package Definition. In the wizard that opens, highlight the package definition you created for Production level: *your_user_id*.eebaw.mortgage.package.prod. Then select **OK**.

__ii. For Load method, make sure **Copy** is selected.

__iii. For the Deployed package root directory, type ${build.directory}/*your_user_id*.eebaw.mortgage.deploy.prod.

__iv. For the Original package root directory, make sure it matches the production level package directory: ${build.directory}/*your_user_id*.eebaw.mortgage.package.prod.

__d. Within the z/OS Deployment tab switch to the Deploy sub-tab:

__i. Modify the copied mappings by replacing the QA qualifier in the PDS names with the PROD qualifier. Use the following table as a reference:

From PDS	To PDS
BUILDER.*YOUR_USER_ID*.PROD.OBJ	BUILDER.*YOUR_USER_ID*.PROD.DEPLOY.OBJ
BUILDER.*YOUR_USER_ID*.PROD.COPYBOOK	BUILDER.*YOUR_USER_ID*.PROD.DEPLOY.COPYBOOK
BUILDER.*YOUR_USER_ID*.PROD.LOAD	BUILDER.*YOUR_USER_ID*.PROD.DEPLOY.LOAD
BUILDER.*YOUR_USER_ID*.PROD.DBRM	BUILDER.*YOUR_USER_ID*.PROD.DEPLOY.DBRM

__e. Optionally specify any publish options available in the Publish sub-tab within the z/OS Deployment tab.

__f. **Save** your changes and close the Deployment Definition editor.

5.4 Summary

In this lab you have performed the necessary configuration to be able to package and deploy your application. You have configured a default ship list for packaging the JKE Banking mortgage application, and explored the main configuration options available for packaging and deploying applications.

The needed configuration is done for the pilot mortgage application. In the next lab you will be able to test all the pieces and validate the configuration with an end-to-end scenario.

Lab 6 Performing an end-to-end verification of your development lifecycle

In this lab, you will perform an end-to-end check of the configuration performed throughout the workshop to adopt the Enterprise Extensions features of Rational Team Concert for build and deployment in z/OS.

After selecting the mortgage application to pilot the adoption of the SCM, build and deploy capabilities of RTC within JKE Banking, you performed a study of how the application is developed and built. Based on that study you have migrated the mortgage application code to Jazz SCM and used the Enterprise Extensions features to configure the build, promote and deploy operations for z/OS. You are now prepared to test a complete cycle of developing and building the mortgage application using these features.

Once you have completed this verification, the team will be ready to adopt the solution and fully use it. This will be the first step toward a complete adoption by JKE Banking z/OS development teams.

In this lab you will:

- Create a change at Development level and build it
- Promote the changed source and built outputs and deploy them to QA

Solution configuration test for adoption

In this lab you will test and validate the configuration steps performed throughout the previous labs of this workshop. When you are adopting RTC build and deploy features in your environment, you will typically want to test the different pieces as you configure them.

6.1 Create a change and build it

In this section, acting as a developer, you will perform a code change at Development level and build it. You will confirm that only your changed program (and any impacted programs) is built.

__1. If it's not still open from the previous lab, start your Rational Team Concert client and open the Work Items perspective

__2. Create a development workspace:

__a. Within the EEBAW JKE Banking project area, expand the Source Control node.

__b. Right-click the **EEBAW JKE Development Stream** and select **New > Repository Workspace.**

__c. Give the repository workspace a name following the workshop convention:
your_user_id Mortgage Development Workspace

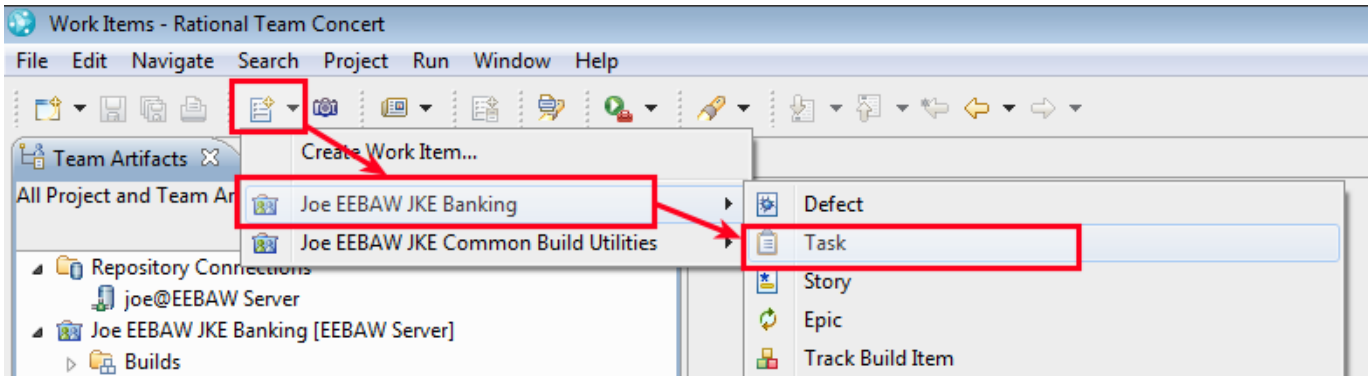
__d. You can click **Finish** so the workspace is created with the defaults.

__e. When the Load Workspace wizard opens, click **Finish** again. If you are warned that loading cannot occur and presented with the Confirm Overwrite, click **Select All** and then **Finish** again.

Asked to overwrite on loading?
You will normally use a different eclipse workspace for the different tasks you perform. In previous labs, you created and loaded a workspace for sharing the code. Now, you are trying to load the same projects from this newly created workspace into the same sandbox. Don't worry... Overwriting your files locally won't overwrite or change anything in the SCM.

__3. Create a task for the change:

__a. Create a new work item of type Task. The screenshot shows one way you can do this:

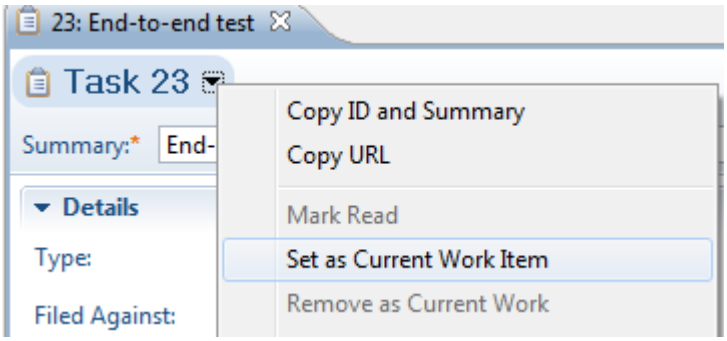


__b. For the **Summary**, type “End-to-end test”.

__c. For the **Planned For** attribute, select the current iteration “Sprint 2”.

__d. You can fill other attributes as you wish, then click **Save**.

__e. Click the work item header and select **Set as Current Work Item**.



__4. Perform a change in the code:

__a. Switch to the Resource perspective and the Project Explorer view.

__b. Open `JKECMORT.cbl` file located at **MortgageApplication-JKECMORT project > zOSsrc > COBOL.**

__c. Add a comment. For example, add the following to the beginning of the program along with the other comments (indicated with an asterisk). Ensure the asterisk begins in column 7.

```
* EEBAW Scenario test
```

__d. **Save** the changes.

__5. Check-in and deliver your changes:

__a. Open the Pending Changes view if it is not already open.

__b. Right-click the **Unresolved** change under Mortgage component and select **Check-in > New Change Set.**

__c. Right-click the checked-in change, select **Edit Comment**, and type “Environment setup test”.

__d. Right-click the checked-in change and select **Deliver.**

__6. Perform a dependency build:

__a. Switch back to the Work Items perspective.

__b. Within the EEBAW JKE Banking project area, expand the Builds node.

__c. Right-click the Development level build with the ID *your_user_id*`.eebaw.mortgage.dev` and select **Request Build**, then **Submit** when the Request Build wizard opens.

__d. The Builds view opens. Click Refresh to get updates to the build progress.

Team and personal dependency builds

The dependency build you have just requested builds from the team stream, and would typically be requested by a team lead or run on a schedule. Individual developers instead would check in a change, request a personal build to build from the files in their personal repository workspace, and then deliver the change to the stream once they are sure their change builds cleanly.

__7. Once the build has completed, check the build results to see what has been built:

__a. Double-click the just completed build in the Builds view to open it.

__b. Switch to the External Links tab.

__c. Click the `buildReport.xml` file link. If asked, log in to the web interface with your workshop user ID and password.

__d. The build report opens. Make sure that just the modified `JKECMORT` file has been built. You will normally see two entries for JKECMORT in the report, reflecting the two outputs that the file build generates along with the reason for building it:

Component name: Joe EEBAW Mortgage						
Project	Source	Reason to build	Code	Language Definition	Output	Max Return Code
MortgageApplication-JKECMORT	BUILDER.JOE.DEV.COBOL (JKECMORT)	1	COB	Joe EEBAW COBOL compilation (CICS&DB2) and link-edit	BUILDER.JOE.DEV.DBRM (JKECMORT)	4
MortgageApplication-JKECMORT	BUILDER.JOE.DEV.COBOL (JKECMORT)	1	COB	Joe EEBAW COBOL compilation (CICS&DB2) and link-edit	BUILDER.JOE.DEV.LOAD (JKECMORT)	4

Useful information in the Downloads tab

In the Downloads tab of the build results, you will find the build report that you just reviewed (buildReport.xml), as well as many artifacts generated and used by the build process. Depending on your build definition configuration and what occurs during the build processing, some of the files may or may not exist for each build result. Some of the useful files you may find are:

"buildableFiles.xml": complete log of the files identified to be built and their dependency information.

"rtcEnterpriseBuild.xml" and *"generatedBuild.xml"*: Ant build files used to perform the build.

"macrodefs.xml": Ant macrodef tasks generated from the system definition contents

You have performed a little change in the development environment and checked that just the modified file is built.

6.2 Promote the change from Development to QA

You will now promote the changes up the hierarchy to the QA level. In this case, you will use the work item promotion feature.

__1. Request promotion from Development to QA:

__a. Within the EEBAW JKE Banking project area, expand the Enterprise Extensions node.

__b. Expand the Promotions node, right-click the promotion with ID *your_user_id*.eebaw.mortgage.promote.devtoqa and select **Promote Work Items**. The Promote Work Items wizard opens.

__c. Make sure **Promote source and outputs** is checked.

Promote source only

By choosing Promote source and outputs, you prevent a re-build of your changes at the target (in this case, QA) level. If a re-build at the target level is desired, choose Promote source only.

__d. Select **Add.**., and select the work item with summary “End-to-end test” that you created in the previous section. Then click **OK.**

__e. (Optional) Preview the elements that will be promoted:

__i. Click **Next** until you reach the Promotion Preview window. This step is not mandatory but allows you to check what is being promoted.

__ii. Click the **Run Preview** button.

__iii. Select the outputs in the preview window and click **View Output** to view the generated report.

__f. Click **Finish**. The promotion begins and the Builds view opens.

__2. The promotion execution generates a summary work item that will open automatically after running the promotion. Review the promotion summary work item:

__a. Check the Overview tab. The Description of the work item gathers general information regarding the work items promoted and the change sets.

__b. Switch to the Links tab. This summary work item is linked to the promoted work items, the change sets that they contained and the promotion build that was executed. This provides you complete traceability.

__3. Review the promotion build result:

__a. From the Builds view, double-click the promotion build result to open it.

__b. In the Downloads tab you can find the `promotionInfo.xml` file which contains a log of the different resources that have been promoted

__c. If you switch to the External Links tab, you will find the build maps that have also been promoted. Clicking them will open the build map information.

You have tested your promotion configuration. You have promoted your test change from Development environment to QA using work item promotion.

6.3 Package and deploy at QA level

We will now package and deploy the promoted change.

__1. Create a work item based package:

__a. Within the EEBAW JKE Banking project area, expand the **Enterprise Extensions > Packages** node.

__b. Right-click *your_user_id*.eebaw.mortgage.package.qa package definition and select **Create Package...** The package creation wizard opens.

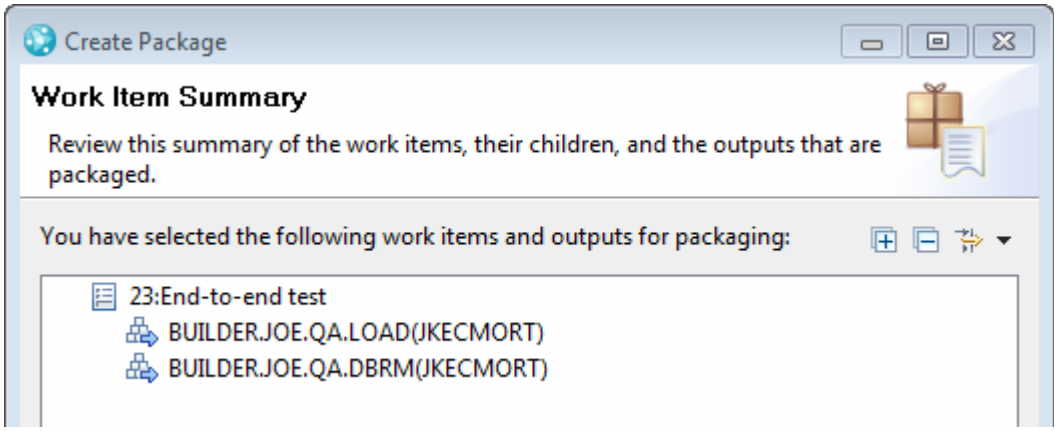
__c. Select **Work Item** packaging and then **Next**.

__d. Similarly to what you did for promoting, click **Add...**

__e. Look for the work item with the summary “End-to-end” that you are using for testing the whole process, highlight it and click **OK**.

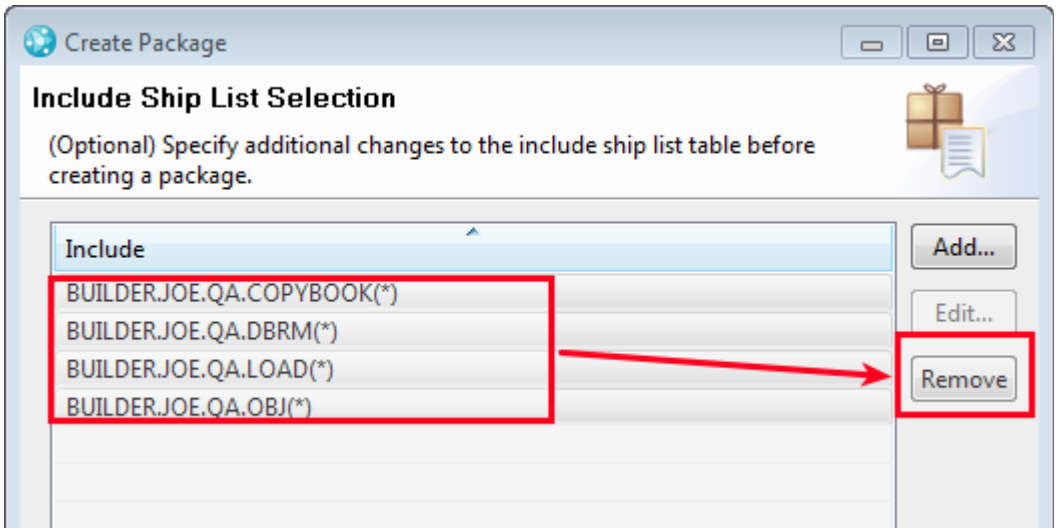
__f. Select **Open summary work item.**

__g. Click **Next.** Inspect the outputs associated with the work item that will be included in the package.



No outputs being showed?
Make sure you are creating your package from the correct QA-level package definition.

__h. Click **Next** until you reach the Include Ship List Selection window. Highlight all of the entries and click **Remove**.



Removing the Ship List entries?

When creating the package definition, you defined a default ship list with the list of the PDS to consider for packaging.

You remove the entries for this package build as you just want to package the outputs from the change promoted using work item promotion. Otherwise all the outputs from the PDS in the ship list definition would be also packaged.

__i. Click **Finish**. You could further customize the package creation but for this test you are accepting the defaults.

__j. The Builds view opens with the progress of the packaging build. Click the refresh button to check how it progresses.

__k. Review the summary work item, which contains information and links similar to the promotion summary work item. Note that the work item is updated with additional information when the package build completes.

__l. Once the package build has completed you can double-click the package build result to open it:

__i. If you chose the corresponding publish options, the Downloads tab contains the package itself along with the manifest.

__ii. In the Package Summary tab you will find information on the members that have been packaged and their PDSs.

__2. Perform a deploy of the package at QA:

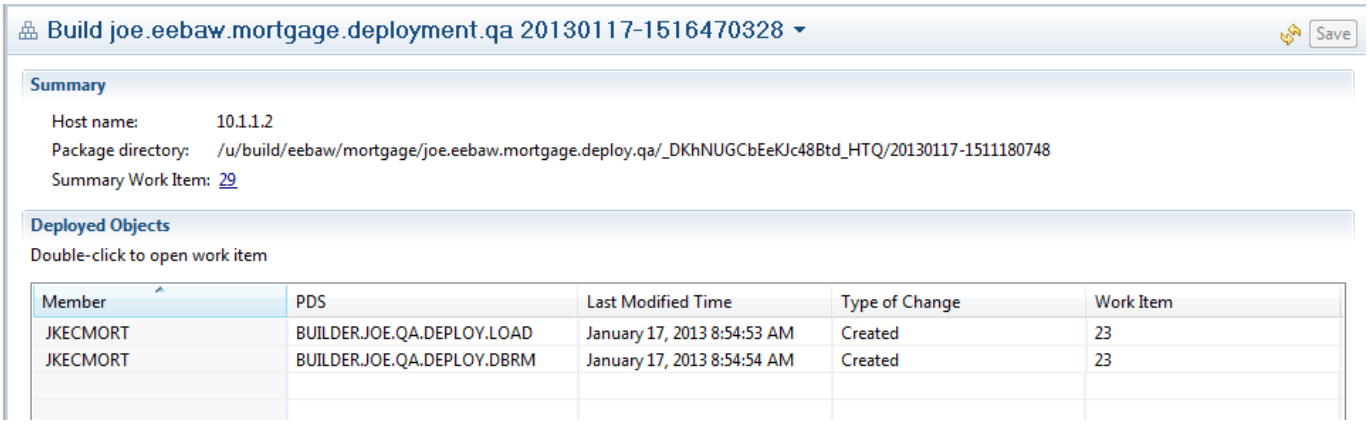
__a. Expand the Deployments node.

__b. Right-click the deployment definition called *your_user_id*.eebaw.mortgage.deployment.qa and select **Request Deployment**.

__c. Accept the defaults and click **Submit**.

__d. The Builds view opens with the progress of the deployment build. Click the refresh button to check how it progresses.

__e. Once the deployment build has completed you can double-click the deployment build result to open it. The Deployment tab gives you a good overview of which members have been deployed and where. Note that the PDSs are the ones you specified in the To PDS column in the restore mapping table in the deployment definition in Lab 5.



__f. A summary work item has been also created with information on the deployed objects.

6.4 Summary

At this point you have tested the configuration of your build, promotion and deployment process implemented in Rational Team Concert. You have tested how individual changes are built using dependency build, and you have promoted your code change to the QA environment using work item based promotion. You have finally packaged and deployed your built changes.

The mortgage application team is ready to work with the solution and you can begin to plan the solution adoption for new JKE Banking teams.

Appendix A Troubleshooting

Lab 1: Planning your Rational Team Concert solution

You may encounter the following errors when running setupCommonSystemDefs.xml:

1. **Failed to create task or type**

   Symptom:

   Console entry:

```
BUILD FAILED

C:\Users\IBM_ADMIN\Workspaces\RDz_85_4xDev_20121205\CommonBuildScripts\
setupCommonSystemDefs.xml:137: Problem: failed to create task or type
antlib:com.ibm.team.enterprise.zos.systemdefinition.toolkit:init

Cause: The name is undefined.
```

   Explanation:

   When you run the Ant script, be sure to choose **Run As>Ant Build...** with the ellipsis points at the end. In the Edit Configuration dialog, you need to switch to the JRE tab and ensure “Run in the same JRE as the workspace” is selected.

2. **NullPointerException**

   Symptom:

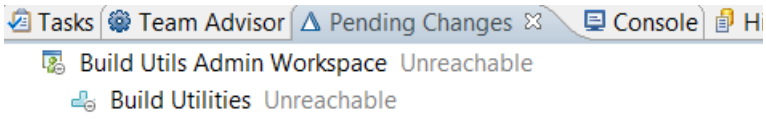
   Console entry:

```
BUILD FAILED

C:\Users\IBM_ADMIN\Workspaces\RDz_85_4xDev_20121205\CommonBuildScripts\setupCom
monSystemDefs.xml:137: java.lang.NullPointerException
```

   Explanation:

   Check your Pending Changes view and ensure that the repository workspace from which you loaded the CommonBuildScripts project is showing as logged in. A label of “Unreachable” will cause the error above.



3. **Existing definitions**

   Symptom:

   Console entry:

```
dsdefs.common:

    [ld:dsdef] Create data set definition (Joe EEBAW BIND files).
    [ld:dsdef] An data set definition with the same name (Joe EEBAW BIND files)
was found in this project area.  It will be updated with new attributes.
     [ld:init] An existing resource definition with the same name (Joe EEBAW
BIND files) was found. It will be updated with the new resource definition.
```

   Explanation:

   Remember that system definitions are repository wide. If you have already run this script once before, it makes sense that you're seeing these messages and you are simply returning all of the system definitions to the values specified in the script. If this is your first time running the script, someone else has used the same prefix as you (or perhaps you both forgot to add a prefix!). Update the script to a new prefix and re-run, and if you are working in a classroom environment, alert your instructor regarding this collision.

Lab 2: Sharing your source members in Rational Team Concert

For issues running setuprtcz.xml, see the troubleshooting entries for running setupCommonSystemDefs.xml in Lab 1 above.

Lab 3: Migrating your build to Rational Team Concert

You may encounter the following errors when running your production dependency build:

1. **Invalid system definition**

   Symptom:

ServerRuntimeError.log entry:

```
com.ibm.team.repository.common.TeamRepositoryException: Invalid system
definition was found. A dependency build cannot be executed.
```

Explanation:

A zFolder or zFile is associated with a system definition that no longer exists. You can check the individual folder and file properties, check the file properties by using the Assign a Language Definition wizard and viewing current assignments, or re-run the setup script to re-associate all of the system definitions.

2. **Illegal read access**

Symptom:

build-*timestamp*.log entry:

```
* fetch:
*
* BUILD FAILED
* com.ibm.team.repository.common.PermissionDeniedException: CRJAZ1316I Illegal
read access:  User "builder2" attempted to read item(s) having the following
type(s): BuildResult
```

Explanation:

The jazz user you have configured your build agent to use does not have permission to read build results in your project area. Make sure your builder ID is a member of your project area and is assigned a role with enough permission to save build results and build engines.

3. **No 'Save Build Result' permission**

Symptom:

build-*timestamp*.log entry:

```
* fetch:
*
* BUILD FAILED
* com.ibm.team.process.common.advice.TeamOperationCanceledException: CRJAZ6053E
The 'Save Build Result' operation cannot be completed. Permission is required
to complete the operation. For more details, open the help system and search
for CRJAZ6053E.
```

Explanation:

See “Illegal read access” explanation above.

4. **Component not found in registry**

Symptom:

build-*timestamp*.log entry:

```
* fetch:
*
* BUILD FAILED
* CRHTC1543E Component (Mortgage Common) is not found in the repository.
```

Explanation:

This message can occur when the component is owned by an individual rather than a process area. Open the stream you are trying to build in the Stream editor and verify ownership in the Components section.

5. **File not found**

Symptom:

build-*timestamp*.log entry:

```
* /u/ryehle/ryehle.eebaw.mortgage.prod/macrodefs.xml:88: CRHTC1664E The file
INSTALL.HHOP850.SFEKLOAD was not found.
*       at
com.ibm.team.enterprise.zos.build.ant.types.Alloc.allocate(Alloc.java:306)
*       at
com.ibm.team.enterprise.zos.build.ant.types.Alloc.allocate(Alloc.java:213)
*       at
com.ibm.team.enterprise.zos.build.ant.types.Concat.allocate(Concat.java:54)
*       at
com.ibm.team.enterprise.zos.build.ant.tasks.Executable.allocateDDs(Executable.j
ava:179)
```

Explanation:

It is likely that one of your *Existing data set used for build* data set definitions has an HLQ that is not valid on your system. Find the offending data set definition based on the name in the error message, and use the RDz Remote Systems view or 3.4 in your favorite terminal emulator to verify the correct HLQ. Update your data set definition accordingly.

6. **Path to non-existent file**

Symptom:

build-*timestamp*.log entry:

```
* fetch:
*
* BUILD FAILED
* /u/build/eebaw/mortgage/joe.eebaw.mortgage.prod/rtcEnterpriseBuild.xml:8:
"passwordFile" contains path to non-existent file "/etc/jazz40/ccm/jazzpw".
*       at
com.ibm.team.build.ant.task.AbstractTeamBuildTask$FileAntAttribute.validate(Abs
tractTeamBuildTask.java:1334)
*       at
com.ibm.team.build.ant.task.AbstractTeamBuildTask.validateAttribute(AbstractTea
mBuildTask.java:853)
*       at
com.ibm.team.build.ant.task.AbstractTeamBuildTask.validateAntTaskAttributes(Abs
tractTeamBuildTask.java:763)
*       at
com.ibm.team.build.ant.task.AbstractTeamBuildTask.execute(AbstractTeamBuildTask
.java:660)
*       at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:288)
```

Explanation:

Make sure that the TSO user ID specified for the Build Agent has the needed access permission to the USS installation configuration files path, like: “`/etc/jazz/ccm`”.

If using the IBM Internal ISDz Appliance, make sure you have specified the “IBMUSER” TSO ID as the user for the Build Agent.

7. **RTCzMVSExec File not found in java.library.path**

Symptom:

build-*timestamp*.log entry:

```
* BUILD FAILED
* java.lang.UnsatisfiedLinkError: RTCzMVSExec (Not found in java.library.path)
*     at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1028)
*     at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:992)
*     at java.lang.System.loadLibrary(System.java:513)
*     at
com.ibm.teamz.build.ant.jni.Executable.loadNativeLibrary(Executable.java:33)
*     at com.ibm.teamz.build.ant.jni.Executable.<clinit>(Executable.java:27)
*     at java.lang.J9VMInternals.initializeImpl(Native Method)
*     at java.lang.J9VMInternals.initialize(J9VMInternals.java:205)
*     at
com.ibm.teamz.build.ant.jni.ISPFStatistics.<clinit>(ISPFStatistics.java:148)
*     at java.lang.J9VMInternals.initializeImpl(Native Method)
```

Explanation:

There may be two possible issues related with this:

- Make sure that the TSO user ID configured for the Build Agent has read and execute permission to the RTCzMVSExec.so file. If not, grant the needed permissions to this file using OMVS

- Make sure that the folder of this “.so” file is loaded in the libpath for the TSO user ID being used for configuring the Build Agent

If using the IBM Internal ISDz Appliance, make sure you have specified the “IBMUSER” TSO ID as the user for the Build Agent.

Additional troubleshooting tips can be found in the Information Center topic Troubleshooting Ant-based z/OS builds.

Lab 4: Promoting your changes from Development to Production

You may encounter the following errors when running the primer component promotions:

1. **Target dependency build fails**

   Symptom:

   The dependency build triggered by the primer component promotion fails with the following message in ServerRuntimeError.log:

   ```
   com.ibm.team.repository.common.TeamRepositoryException:
   com.ibm.team.build.common.TeamBuildException: The specified build workspace
   contains no buildable programs. A build.xml file cannot be generated.
   ```

   Explanation:

   You are hitting a known bug, Promote Component is successful, but target dependency build definition shows failure (231295). You can simply request another dependency build at the target level to clear up this error.

2. **Primer Promote fails with: CRJZZ0557E: Failed running Copy**

   Symptom:

   build-*timestamp*.log entry:

   ```
   * BUILD FAILED

   * /u/build/eebaw/mortgage/joe.eebaw.mortgage.primer/generatedBuild.xml:29:
   CRJZZ0557E: Failed running Copy. Look at the CDATA section above for the
   failure reason. Check for &lt;REASON-CODE&gt; entries if they exist.

   Further checking the CDATA section, contains:
   *      [exec]   <![CDATA[
   *      [exec]    IRX0157E Routine EAGRTXIN of the run time processor EAGRTPRC
   was not found.
   *      [exec]    ISPD117
   *      [exec]    The initially invoked CLIST ended with a return code = 20
   *      [exec]   ]]>
   *      [exec]   </ISPF>
   ```

   Explanation:

Check your ISPF Gateway configuration. Some of the modules specified in the configuration file are not available in the ISPF concatenations. Check the configuration in “ispf.conf” file and the documentation in the following InfoCenter entry:

http://pic.dhe.ibm.com/infocenter/clmhelp/v4r0/index.jsp?topic=%2Fcom.ibm.jazz.install.doc%2Ftopics%2Fc_rtcz_customize_ispfgateway.html

If using the IBM Internal ISDz Appliance, make sure you have **not** IPLed the z/OS server with option 1: “ISPFPROC. Basic IPL without DB2, etc. Cold start”

Lab 5: Packaging and deploying your application

Lab 6: Performing an end-to-end verification of your development lifecycle

Appendix B Notices

This information was developed for products and services offered in the U.S.A.

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

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

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

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

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

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix C Trademarks and copyrights

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

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	Tivoli	WebSphere	Workplace	System p	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

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

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES

NOTES

Please Recycle

IBM Software