

Type of Submission: Tutorial

Title: Jazz Tutorial – Using RTC for a group project

Subtitle: An Introduction to Teams, Work Items, Source Control and Iteration Plans in Rational Team Concert.

Keywords: work item, team, source control, component, iteration plan, sprint, backlog

Given: Thomas

Family: Fritz

Job Title: Postdoctoral Researcher

Email: fritz@cs.ubc.ca

Bio: Thomas Fritz received the Diplom degree in computer science from the Ludwig-Maximilians-University Munich, Germany, in 2005. He received the PhD degree in computer science from the University of British Columbia in 2011. He has experience working as an intern with several companies including the IBM OTI labs in Zurich and Ottawa where he worked with the Jazz work item and source control teams. He has taught the undergraduate introductory course to software engineering in the department of computer science at UBC and is currently a postdoctoral researcher at UBC. His research focuses on how to help software developers better manage the information and systems on which they work.

Company: University of British Columbia



Given: Meghan

Family: Allen

Job Title: Lecturer

Email: meghana@cs.ubc.ca

Bio: Meghan Allen received her BSc and MSc degrees in computer science from the University of British Columbia (UBC) in 2001 and 2006. She has experience working as a software developer where she used team coordination tools "in the wild". Since 2007 she has been a lecturer for the department of computer science at UBC where she has taught a variety of undergraduate courses including the introductory course in software engineering which uses RTC.

Company: University of British Columbia



Abstract:

Software engineering comprises activities such as breaking down a development project into manageable tasks, creating and changing source code, communicating with fellow developers and managing teams. The Rational Team Concert tool (RTC) supports many software engineering activities and thus can form the cornerstone of your project development. In the following, we will explore some of the major concepts of RTC, such as work items, teams, source control, and iteration plans, and how these concepts can be used in developing a small software project using an agile process in a class setting.

Jazz Tutorial – Using RTC for a group project – Part I

Introduction

The Jazz platform, and the associated Rational Team Concert tool (RTC), can form the cornerstone of your project development. This tool embodies a modern approach to software engineering and, as such, is a very good way to learn, hands-on, the important concepts of software engineering.

Although you might be familiar with a lot of the concepts present in RTC, learning to use a new software tool is never easy. For example, the specific nomenclature of RTC will sometime differ from the one you are used to.

The goal of this tutorial is to introduce Jazz and RTC by focusing on a small class project. It is in no way a complete documentation of the tool, and at some point you will probably need to dig deeper in order to find answers to your questions.

Going beyond this tutorial

There are lots of ways to learn more about Jazz and RTC. A lot of additional resources are available on the Jazz website. Some of them require that you created an account. To check the documentation, just go to <http://jazz.net>, login and select the *Learn* link from the menu. For documentation about Eclipse, check <http://eclipse.org>.

In case you have looked around and are still unable to find an answer to your question, you can always post a question on one of the news forums on <http://jazz.net>.

Following this tutorial

Pay particular attention to everything that is bold. It either describes a very important concept or an action you need to take in order to successfully complete this tutorial.

1 - Installation and setup

Introduction

Chapter 1 of this tutorial will guide you through the process of setting up and running RTC for the first time.

RTC and Eclipse

Eclipse, which you may have used before, is a very good development platform for your Java projects. However, Eclipse is more than a code editor; it's an entirely open platform that can be extended by third-party developers. Rational Team Concert is exactly that: an Eclipse expansion provided by IBM Rational in order to facilitate and coordinate team work.

As opposed to typical Eclipse plug-ins, however, RTC includes so many additions and customizations that it cannot be installed on top of your current version of Eclipse. Instead, you will need to download and install it from the IBM Jazz website and what you will get is a complete bundle that includes Eclipse, the typical Java development plug-ins, and all the plug-ins required for RTC to run.

Extending and upgrading your new Eclipse

Once you get your new Eclipse (RTC), you may be tempted to extend it with all your favourite plug-ins, or to upgrade it to the latest version. Although this can be done, it is not as simple as with a standard Eclipse installation because IBM uses customized versions of various standard packages. These customized packages live in a different directory and cannot be overridden. In other words, extending or upgrading RTC's version of Eclipse is not as easy as extending or upgrading Eclipse. In all cases, the bundled version of Eclipse has all the tools we will need in this tutorial.

Installing RTC on your own machine

This section will show you how to download and install Rational Team Concert on your home PC or your laptop. We strongly recommend that you do this since it will allow you to participate on the project from home. It will also let you enjoy one great aspect of source control, and of RTC in particular, namely the fact that it can automatically replicate your entire development environment across the various computers you use. In all cases, you will need to join the Jazz program if you want to be able to browse the online documentation and download the RTC client.

Joining the Jazz program

If you want to download Rational Team Concert, or even if you only want to browse the associated documentation, then **you will need to register on the Jazz Community Site**. To do this, go to <http://jazz.net> and click the "Register now!" button on the right-hand side and follow the entire procedure.

Downloading the Jazz client (version 2.0.0.2iFix5)

Once you have registered and logged in, point your browser to the following download page: <https://jazz.net/downloads/rational-team-concert/releases/2.0.0.2iFix5> (in case you are connecting to a server that has a newer version installed, make sure to download a client version that is compatible).

The jazz download pages have a history of moving around, so if the link does not work, start from <http://jazz.net>, login and follow the “downloads” link on the right side.

Make sure that you are on the page for the version of Rational Team Concert that is compatible with your server (in our case 2.0.0.2 iFix5). You **must not download any other version** of RTC as it might not be compatible with the Jazz server.

Select from Windows or Linux from the list just under “Features” on the left hand side of the page. Then, select “Download” for Express-C. You can then accept the license agreement and your download should start. If you use a Mac, I think you want the “Client for Eclipse IDE” that is listed on the “All Downloads” page under “Incubators”. Again, make sure you’re still on the Rational Team Concert site of the version your server is compatible with, since newer versions won’t necessarily be consistent with your server.

Installing the Jazz client

I have never installed the client on a Mac, so you will have to figure this part out by yourself. On a Windows platform simply unzip the file you just downloaded.

Running RTC for the first time

To run RTC, simply go to the folder on your windows machine that you just unzipped and then open `jazz\client\eclipse` and double click on the `eclipse.exe`.

Selecting a workspace

The first thing RTC will ask you upon start-up is to provide a workspace. The concept of workspace is part of Eclipse, it indicates the directory where all of your source files, binaries, external jars and other development resources (such as icons) will be stored.

This is intended to be the unique entry point for development, so it may eventually get filled with various unrelated projects and libraries. This is not really a problem, however, since Eclipse offers various techniques to efficiently filter your workspace content.

If you already have an Eclipse workspace, **you should not use it with RTC**. This is because RTC uses a customized version of Eclipse that seems to create various incompatibilities. You should therefore make sure you are creating a **new workspace** specifically for RTC.

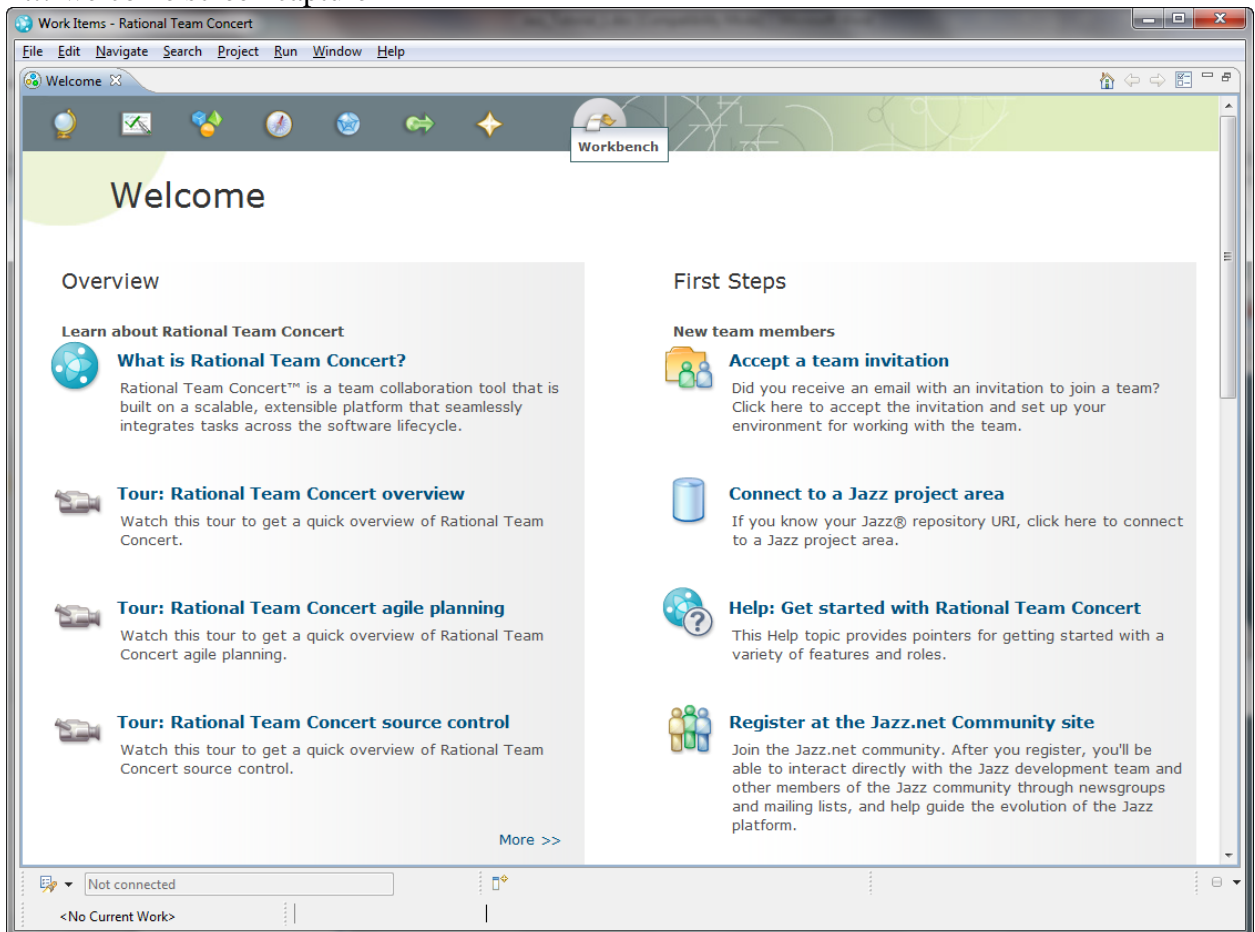
Welcome screen

After configuring your workspace, you should see the following welcome screen.

Tip: If you don’t get this welcome screen, or if you want to return to it later, simply select *Help > Welcome* from the RTC menu.

Figure 1. Welcome screen for the RTC client.

Alt: welcome screen capture



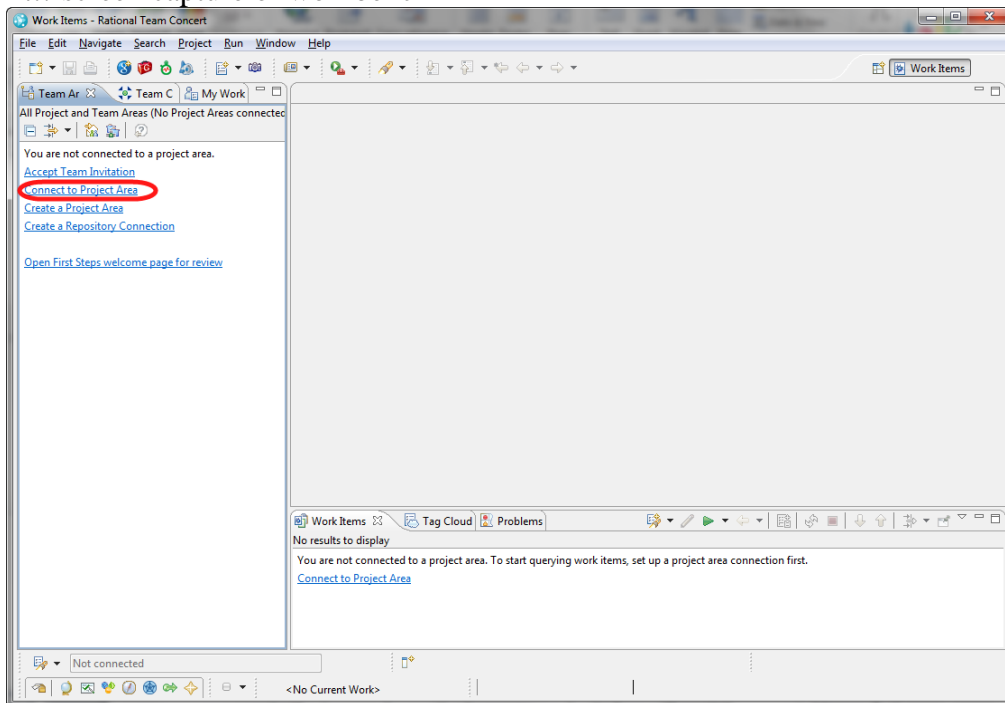
Click on *Workbench*.

Eclipse Views and Perspective

The **Workbench** refers to the main Eclipse window and, for now, it should look similar to this:

Figure 2. Workbench with Team Artifacts View open.

Alt: screen capture of workbench



The workbench contains a menu, a toolbar, a status bar, and an arrangement of subwindows. Each of these subwindows contains a number of Tabs. For example the currently selected tab on the left is called “Team Artifacts”. In Eclipse, the content of these tabs are called **Views** and are used to gather together related information.

In the image above (Figure 2), the large gray subwindow will contain a special type of views called **Editors**. You probably already know the standard editor that you use for entering Java code. Other kinds of editors we will meet include the Work Item editor and the Iteration Plan editor, among others.

Notice on the upper right the button “Work Items”. This button indicates the current Eclipse **Perspective**. A Perspective is simply a specific configuration of Views and Editors that facilitates a particular task. When working with RTC, you usually want the *Work Items* perspective. When editing code, you will want to be in the *Java* perspective. When debugging, you will use the *Debug* perspective. You can change perspective through the menu *Window > Open Perspective*.

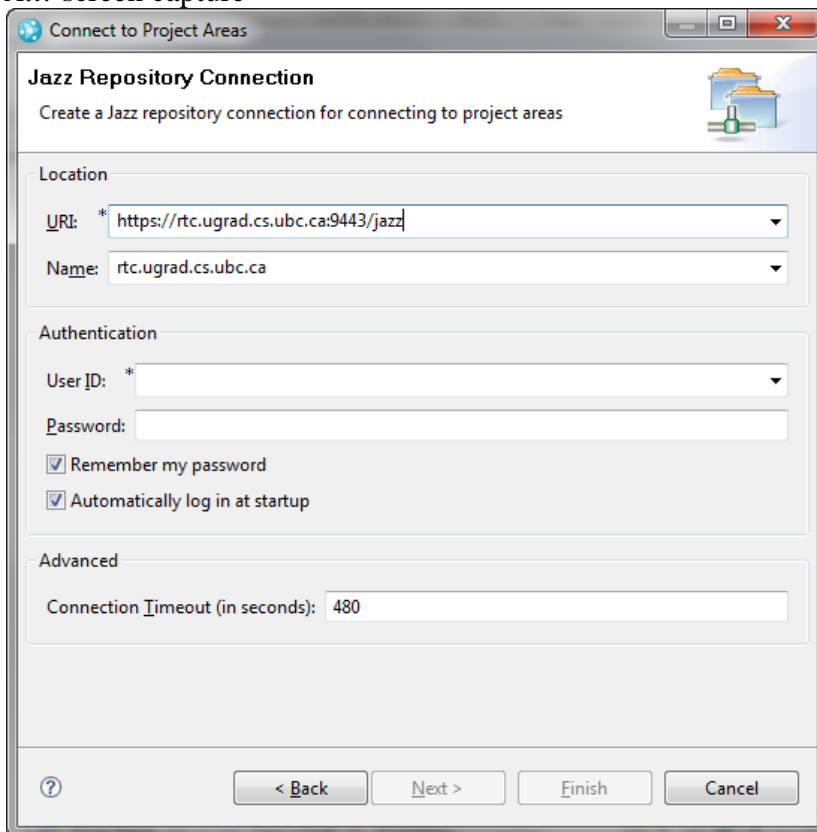
Remember that a perspective is simply a useful arrangement of views. All the views can be accessed in all the perspectives using *Window > Show View*. Views can also be moved around, reorganized or closed by clicking the “X” button in the tab. If you mess too much with your workbench, you can always clean it up using *Window > Reset Perspective*.

Connecting to a repository

You will need to manually **connect to a repository**. To do so, make sure you are in the *Work Items* perspective and that the *Team Artifact* view is selected. Then click on the “Connect to project area” link. This should pop-up a dialog. Select “Create a new repository connection” and click Next.

Figure 3. Wizard for connecting to a Project Area.

Alt: screen capture



The screenshot shows a Windows-style dialog box titled "Connect to Project Areas". The main heading is "Jazz Repository Connection" with a sub-heading "Create a Jazz repository connection for connecting to project areas". The dialog is divided into three sections: "Location", "Authentication", and "Advanced".

- Location:** Contains two dropdown menus. The "URI:" dropdown is set to "https://rtc.ugrad.cs.ubc.ca:9443/jazz". The "Name:" dropdown is set to "rtc.ugrad.cs.ubc.ca".
- Authentication:** Contains a "User ID:" dropdown menu, a "Password:" text input field, and two checked checkboxes: "Remember my password" and "Automatically log in at startup".
- Advanced:** Contains a "Connection Timeout (in seconds):" text input field set to "480".

At the bottom of the dialog, there are four buttons: a help icon (?), "< Back", "Next >", "Finish", and "Cancel".

You should then fill-out the form as indicated above, but enter your server URI and your own User ID and Password. Click Next to connect to the Jazz server.

In case the connection fails, make sure you typed the correct URI. Second, check the User ID and password. Finally, make sure you are using the right client version.

You will then see a list of **Project Areas**.

Figure 4. Exemplary list of Project Areas.

Alt: screen capture of Project Area selection list.

- CPSC 310 - Fall 10
- CPSC 310 - Spring 11
- CPSC 410 - Team 1
- CPSC 410 - Team 2
- CPSC 410 - Team 3
- CPSC 410 - Team 4
- CPSC 410 - Team 5
- CPSC 410 - Team 6
- Sandbox
- Test

You should select the main project area you will be using. In our case, we select *CPSC 310 – Spring 11*, the Project Area for the class project we are working with. In addition, you can select other project areas. Click “Finish” and you will see, in the *Team Artifact* view, that your repository connection has been created, together with the selected project areas.

Note that there is a difference between Java Projects and RTC’s Project Areas. A project area is expected to group together all the development effort related to a specific product, but it is often the case that a product contains numerous Java projects.

2 – Teams

Introduction

You should already be added to a team by the server administrator before you continue. You can check if you have been correctly added to your team by looking at “My Team Areas” in the “Team Artifacts” view.

Working in your own account

You should always be working from your own account. Do not edit or update anything using somebody else’s account. It’s important that everything you do and everything you work on be correctly attributed to your user name.

Do not fool around in the main project

You have a lot of permissions in the main project. For example, you could add yourself to more than one team, or try customizing the process for your team. **Please don’t do that** since it could seriously affect the project configuration. If you are interested in exploring various other features of RTC, feel free to create a sandbox Project Area and do so in there.

3 – Work Items

Introduction

A large development project is made of an accumulation of small tasks: from the classes you should code to the bugs you must squash, including even writing the documentation your client requests... These tasks are distributed among the team's members, and it is often very useful to know what the other members are (or *should be*) working on. Since Rational Team Concert is all about organizing team work, it provides very efficient tools to do just that.

Creating and managing Work Items

The cornerstone of team work management, in RTC, is called the **Work Item**. Work Items are (usually short) tasks that need to be performed in order to achieve a specific goal. The many attributes present in a work item lets you add various details, including how long it should take to resolve, who should be working on it and when it is due.

Creating a Work Item

As part of this tutorial, **you are required to create a first work item**. Since this is one of your *tasks*, you should as well create a work item for it! To do this, click the “New work item” (Figure 5). Select the project area you will be working with, in our case the “CPSC 310 – Spring 11” project, and then “Task” for your work item type. You should now see the *Work Item Editor* (Figure 6).

Figure 5. New Work Item button.

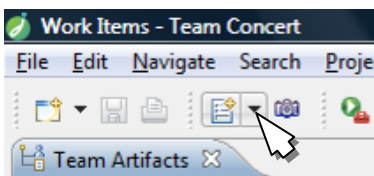
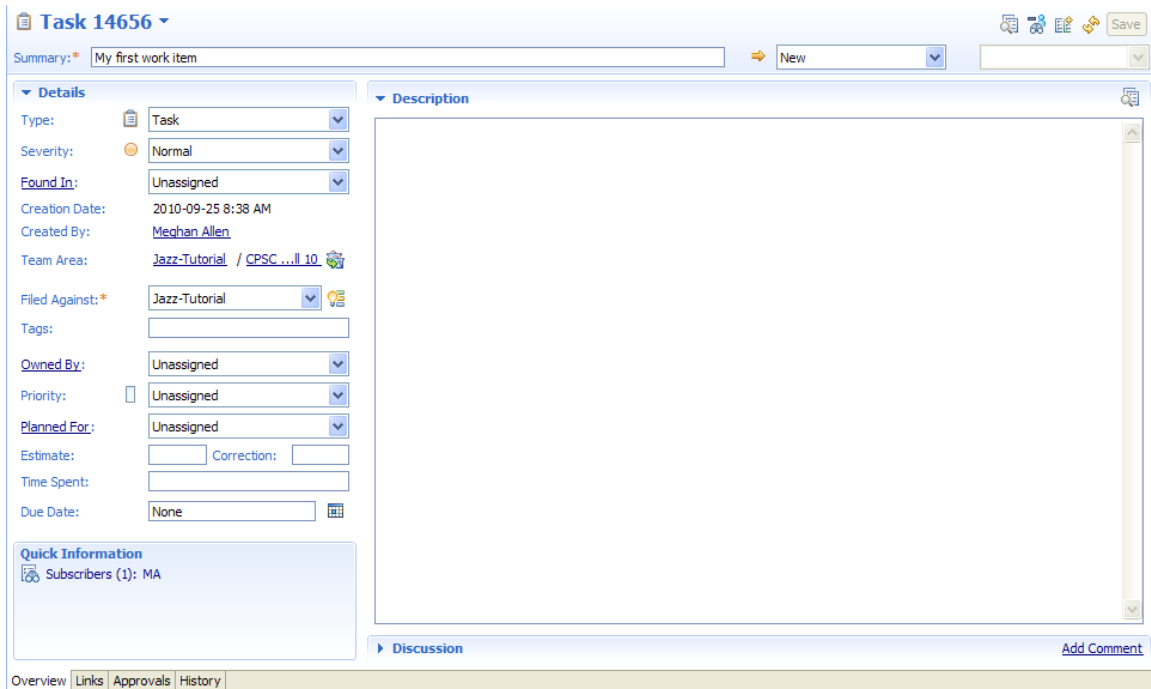


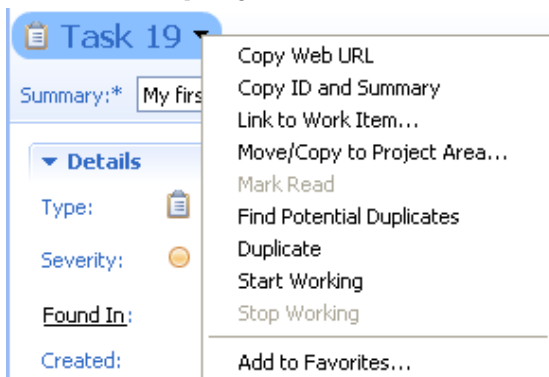
Figure 6. Work Item Editor.



You need to fill out this task. Every work item **must have a Summary**, so enter one (For example: “My first Work Item”). Another field **you need to fill out for every work item is Filed Against**. In our case, we have a category “Jazz Tutorial” for this tutorial that we select. **Another field you should fill in every Work Item is the Planned For** attribute. This describes which iteration the Work Item is planned for and makes it appear in the Iteration Plan, which we’ll describe shortly. We choose Jazz Tutorial Iteration, the iteration that was created for this tutorial. Finally, in the *Owned By* field, you can assign the work item to any member of your team. For now, **assign the work item to yourself**.

If you don't see the category that you are looking for, there can be various reasons. Most likely, you did not add the Work Item to the right project area. Click the little arrow on the right of the Work Item's title then select Move/Copy to Project Area (Figure 7)...

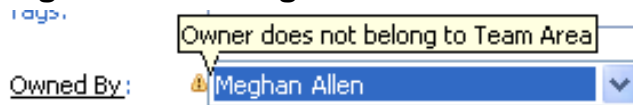
Figure 7. Task drop down menu allowing you to move/copy a task to a different project area.



You can fill the other fields if you like, however a lot of Work Items do not need such precision. If you want to estimate the required time to complete this task enter something like “5 mins” in the *Estimate* field. If you think the work item is complex enough, add some details in the *Description* field. You get the idea.

It could happen that a warning sign appears. Leaving your mouse on this sign you will probably display “Owner does not belong to Team Area” (see Figure 8). This usually means that you selected the wrong *Category*. Make sure you select a *Category* (in Filed Against) corresponding to the team the desired Owner is in.

Figure 8. Warning about owner of the task.



During the project, you will constantly be creating new Work Items. So adding a Work Item must be quick and easy. Make sure you master the process described above and don’t hesitate to add simple Work Items with the **minimal set of attributes**: *Summary*, *Type*, *Category* and *Planned For*. They can always be clarified later.

Tip: As you noticed, saving the Work Item does not close the Editor. You could manually close it by clicking the “X” button, but sometimes it is good to leave it around so that you can easily come back once you’ve completed the task.

Changing the status of a Work Item

When you are done entering the details of your Work Item, click the “Save” button on the upper-right corner of the Work Item Editor. Notice how the Work Item status changes from “Uninitialized” to “New”. This indicates that the Work Item has been successfully created. Most of the time, you will leave it in that state for now.

When you start working on a Work Item, however, you may want to change its status to indicate this. To do so, click on the drop-down arrow and select “Start Working”. **Do this now for your current Work Item**, then click Save again. The status has changed to “In Progress”.

Don’t worry if you forget to change the status of a task before you start working on it. This status is usually useful for larger tasks. It is often the case that a small task will directly go from “New” to “Resolved”.

Discussing a Work Item

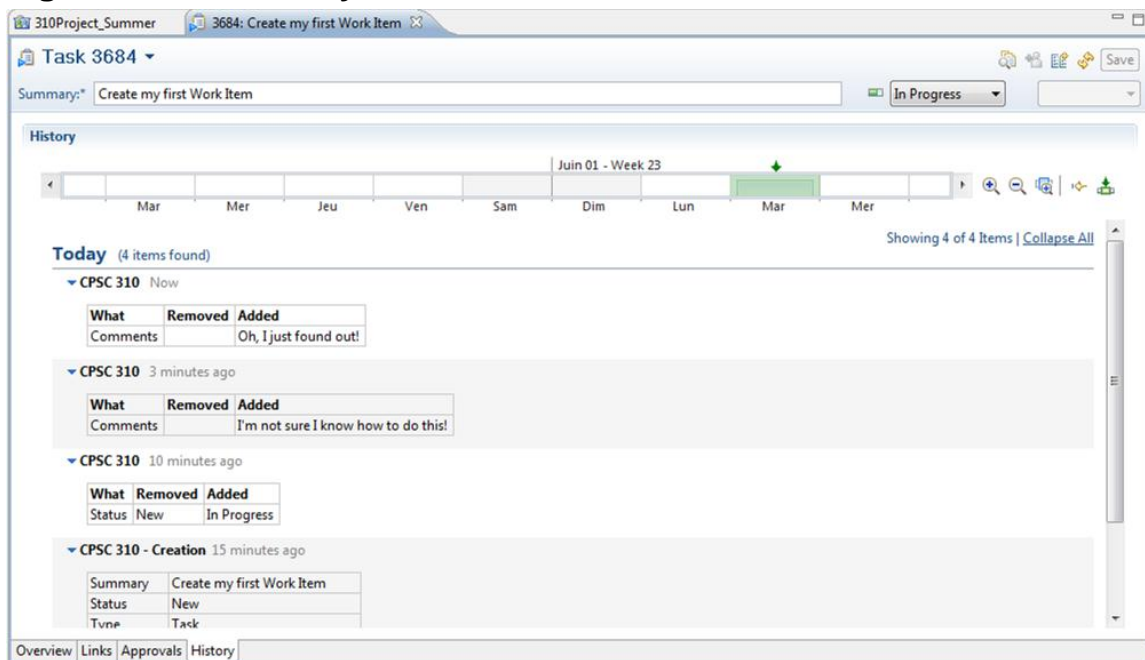
Work items often concern an entire team, and various team members may have an opinion on how to perform the task. The best way to make sure your thoughts are not lost is to record them in the discussion related to the task. To do so, click the “Add Comment”

link at the bottom right of the Work Item editor and enter a comment in the newly created edit box. **For this tutorial you must add at least one comment to the discussion.** When you're done, click Save.

Checking the evolution of a Work Item

Complex tasks will sometimes stay open for a while. When this happens, it is sometime interesting to check what has happened with it in the past. This is achieved by clicking the "History" tab at the bottom of the Work Item Editor. If you click on it now you should see something as presented in Figure 9.

Figure 9.Task History.



This window shows a time-line indicating when activity has been recorded on the Work Item, together with a list of the changes that have occurred. In practice, this kind of view is often used by managers to check the progress of a project and to make sure the workload is well distributed among the team members and that everybody contributes.

Closing your work item

Since you have successfully created this work item you have accomplished your task! Change its status from "In Progress" to "Resolve" then click "Save".

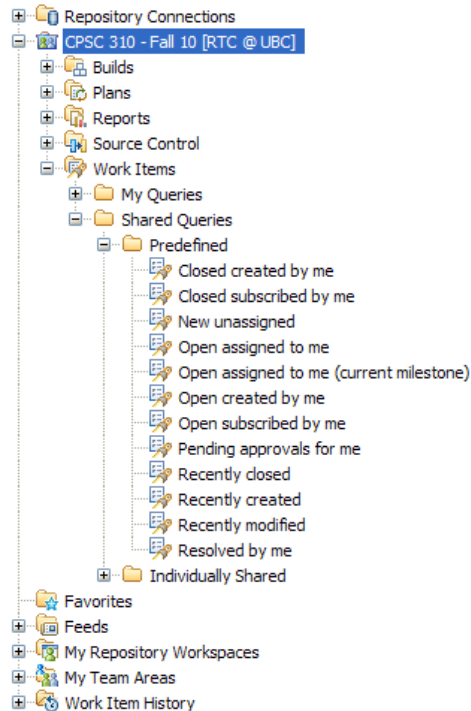
Advanced features

Work items can become quite complex. They can be organized hierarchically, they can refer to one another, they can require approval by other team members, etc. We shouldn't be needing these advanced features, but it is good to know they exist. Feel free to experiment with them in a sandbox project area.

Searching for work items

Creating work items wouldn't be very useful if you were not able to look for them. To search for work items, go to the *Team Artifacts* view, expand the project area, in our case *CPSC 310 - Fall 11*, then *Work Items*, then *Shared Queries* and *Predefined* (Figure 10).

Figure 10. Predefined Queries in the Team Artifacts view.



You should now see a long list of queries. All of these refer to work items and can be useful at various stages of the project development. **Double click on the “Closed created by me” query.**

This will populate the Work Items view, at the bottom of the screen. There you should see that “My first Work Item” is now resolved.

Other queries

Take a look at the other queries. They have descriptive names that let you guess what they are doing. In reality, they are just filters over the various attributes of a Work Item. Try to imagine scenarios in which you would need to know all the Work Items “Open created by me”, or the “New unassigned” work items.

To know exactly what a query is doing, try right-clicking it and selecting *Edit...*

How to use work items

When to create work items

You should create work items as soon as you think of something to do, even if you start doing it immediately! The reason is that work items are not only your *To Do* list, they are an integral part of the project and make it possible to trace its evolution through the accumulation of tasks. So if you start working on some piece of code or writing a design document, make sure you have a corresponding work item!

Tip: You don't have to go to the *Work Items* perspective to add a work item. The "New Work Item" button appears on the toolbar even when you're coding or debugging.

What is a good work item

Let's take an example. Among the following which is the best work item: "implement vector addition", "write the vector class" or "create the mathematics module"? The answer is very subjective: a work item should be small enough to be tackled by a single individual in a short amount of time and it should be descriptive of the work to be performed. However, it shouldn't divide the work to the point that Work Item maintenance becomes tedious. In all cases, don't worry if you don't get the Work Item right on the first try, you can always divide it up later.

4 – Users

Introduction

Teams are made up of users: you and your teammates. This chapter describes how you can find and setup your user page to give a bit more information about you.

Opening a user page

To get more information about a user (or yourself!), you must open the user page. To do so, open the *Team Artifacts* view and expand *My Team Areas*. This will display the list of teams you're a part of – look for your team. Expand this team and you should see the name of all your teammates. **Double-click on your name**, this will bring up the User Editor.

If you want to open the user page of somebody who is not in your team, open the *Process* perspective and from the *Team Organization* view, look for a team this user is in.

Editing a user page

Changing your profile picture

Using this editor you can change various things in your profile, such as your name or your picture. **Change your picture now** by clicking “Browse...” and selecting a local image file. If you don’t have a picture of you, download a small image off the internet and use it as a placeholder. Don’t forget to click Save!

You should try to use a picture of yourself instead of an avatar. This will help your teammates identify you, especially in bigger project teams.

Editing your Working Hours

From your User Editor page, click the Work Environment tab at the bottom. Make sure your Time Zone is set to your local time zone, in our case “America/Vancouver”.

5 – Conclusion

What you should have learned

After completing this tutorial, you should have a basic understanding of the following important concepts of Eclipse and RTC:

- What the Eclipse Perspectives and Eclipse Views are and how to manipulate them;
- That RTC helps organize the *work* of *teams* working on the same *software project*;
- What a Project Area is and how it differs from a standard Java project;
- Why it is important to work from your own account when using RTC;
- What is a Work Item and what are its important attributes;
- How to quickly create a new Work Item and change its status;
- How to discuss a Work Item and check its evolution;
- How to search for work items based on various criteria;
- When you should create Work Items and what you should put in them;
- How to edit your user profile;

What you should have done

If you followed this tutorial correctly, you should have accomplished at least the following:

- Created an account on the Jazz Community Site;
- Launched RTC and created your own workspace;
- Connected to a repository and a project area;
- Created a first work item and changed it status to “Start Working” and then “Resolve”;
- Posted your user picture (or a placeholder);

There is more to RTC

It will definitely be useful to spend some time using RTC *before* the project phase starts. For the project, we will need to introduce new concepts, such as hierarchical teams, streams and source control. This will be explained in a future tutorial that you will work on during a future lab.

For more information visit the website <http://jazz.net> or look at the Help under *Help > Help Contents*. There are several good tutorials/lessons on the usage of basic features of RTC for example under *Tutorials > Do and Learn > Get started with Rational Team Concert*. Get familiar with RTC as soon as possible so that you can concentrate on the actual project later on.

Jazz Tutorial – Using RTC for a group project – Part II

Introduction

The first part of the Jazz tutorial introduced you to the fundamental concepts of Rational Team Concert. It mostly focused on team creation and work items. In this part of the tutorial we will show how RTC can also be used to keep track of the evolution of the project itself through its source control mechanism and a defined software development process with iterations.

Following this tutorial

This document assumes that you read, understood and followed *Part I* of this tutorial. You should also launch RTC and connect to the server before you start this tutorial. **Pay particular attention to everything that is underlined.** It either describes a very important concept or an action you need to take in order to successfully complete this tutorial.

Going further

For more information about source control in RTC you can check the online tutorial:

<https://jazz.net/learn/LearnItem.jsp?href=content/docs/source-control/index.html>

This tutorial digs more deeply into some advanced concepts, such as resolving conflicts.

1 – Overview.

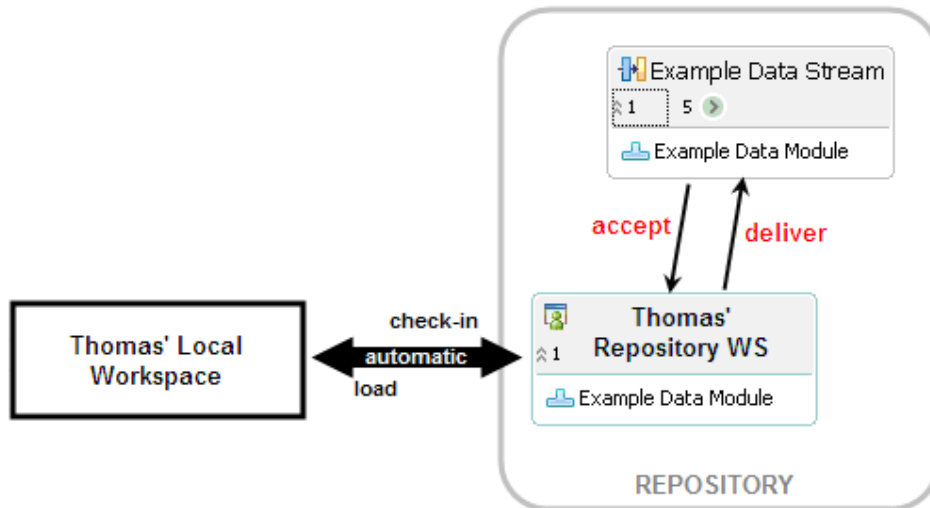
Each student (in our example in Figure 11: *Thomas*), will have his/her own **local workspace** the same way as you have a workspace in Eclipse on your local machine. Each student will also have a **repository workspace** (in Figure 11: Thomas's Repository WS) on the server that serves as a safe copy for all the project code that he/she has checked in.

If you are familiar with CVS, you can think of your repository workspace as your very own CVS repository, into which only you will be checking in code.

Apart from your own repository workspace, you also want to share code with your teammates. For this purpose there is a **data stream** for each team on the server. Once you change the code of the project and you want to save the changes, you should check-in to your repository workspace. Once you're ready to share these changes with the team so that everybody can see them, you just **deliver** them to the stream. Once delivered, your teammates will see that there are incoming changes and they can **accept** them into their workspace. When they accept them, your changes will be in their

repository workspace as well as their local workspace. So the stream serves to share and save your team's code.

Figure 1. Basic Workspace and Stream setup for a project.



2 – Data Streams

Introduction

Before you can start using Jazz source control you need to create a data stream. Remember: a data stream is used by a team to share source code. You will therefore need **only one stream for your entire development team.**

Creating your data stream

You will have to create ONE stream for your development team. So, the first team member to complete this part of the tutorial will have to create the stream. To check if it's already been done, make sure you are using the Work Items perspective and then select the Team Artifacts view and expand *Source Control* under the project area item, in our case *CPSC 310 - Spring 11 > Source Control*. If you see a stream for your team, it's already been created and you can skip to the "Repository Workspace" section (section 3). **If the stream doesn't exist yet, please follow the rest of the instructions in this section to create it.**

Go to the Work Items perspective, open the Team Artifacts view and expand your project area (*CPSC 310 – Spring 11* in our case). **Then right-click on the *Source Control* node and select *New > Stream...*** . This will bring up a Stream editor. In the *Name* box write your stream name. Then click the *Browse* button and select your development team. You can also add a short description of this stream.

You then need to add a component to this stream. In our case, each development team will just use one component, but your component can contain multiple java projects. To add the components click *New...* in the *Components* section and enter a name for your component (“Hockey Pool” or something similar for our hockey pool project) and click *OK*.

Click *Save* to save this stream.

3 – Repository Workspace

Introduction

Before you start using Jazz’s source control you will need to setup your repository workspace. Remember: the repository workspace lives on the Jazz server and acts as a backup copy of your local workspace.

In fact, it is more than a simple *backup copy*, it is a version controlled copy. It keeps track of all the changes you made and you can always revert to an earlier version if things go wrong.

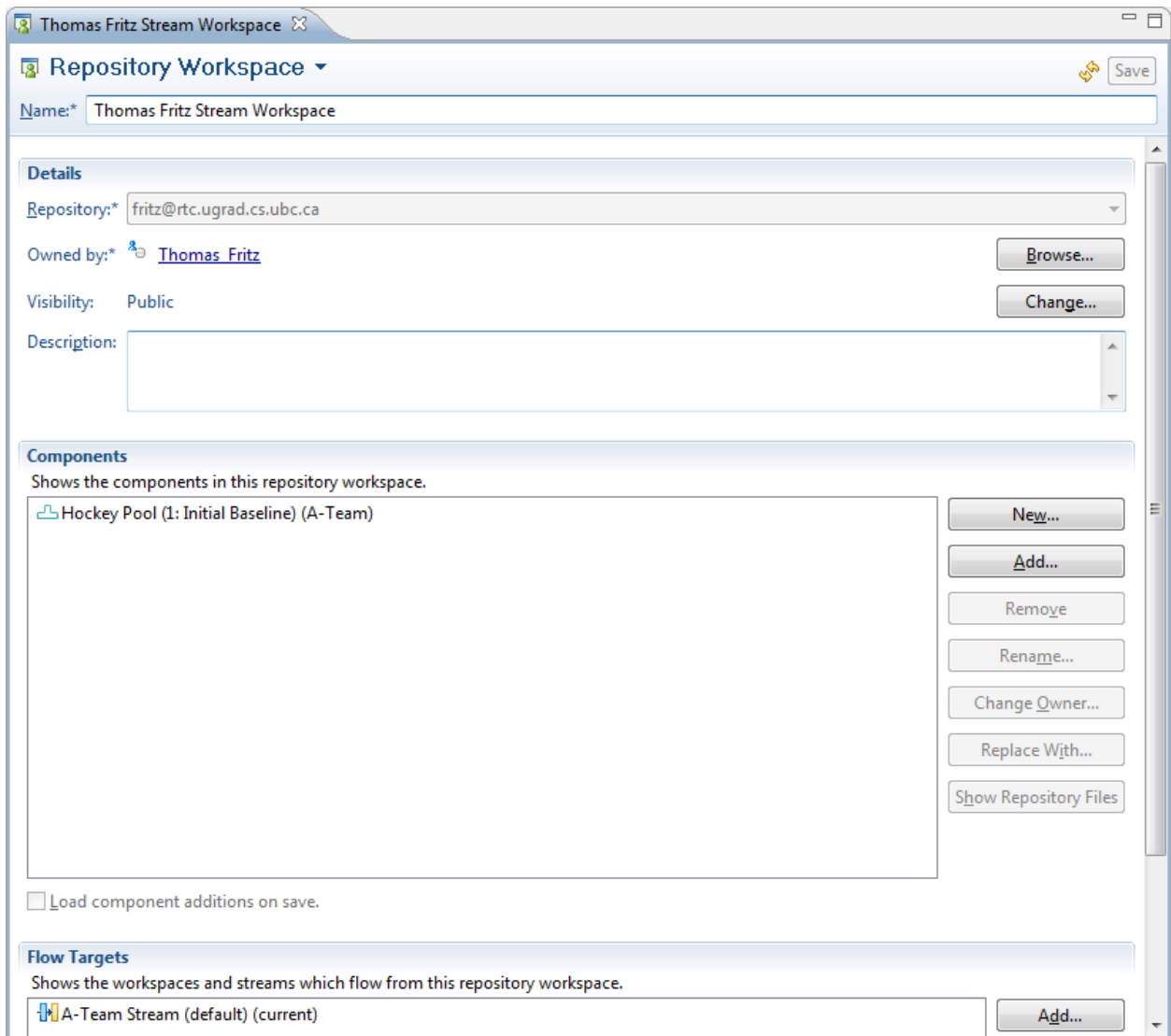
Creating a repository workspace

Since everybody should have his or her own repository workspace, **you need to complete this step individually**. First, make sure you are using the Work Items perspective. From there, select the Team Artifacts view and expand the *Source Control* node under the project area node, in our case *CPSC 310 - Spring 11 > Source Control*. Locate your stream, **right-click on it and select *New Repository Workspace....*** In the repository workspace name enter “*John Doe Repository Workspace*” (replace John Doe by your own name, naturally!) Click *Next*, select *Public*, click *Next* again, make sure the component created in the previous section is selected and click *Finish*.

If you get a “Load Repository Workspace” wizard, you can select “*Find and load Eclipse Projects*”, then click *Finish*.

In the Team Artifacts view you can now expand the *My Repository Workspaces* and you should see your newly created repository workspace. Double-click on it and you should get the following (except that your flow target should be your team’s stream):

Figure 2. Workspace editor.



4 – Using RTC source control

Introduction

Your environment is now correctly setup to use source control. This section of the tutorial will show you how to use source control to backup and exchange your code.

Creating the project

One person on your team needs to create the project that you will be working on. Switch to the Java Perspective and open the Package Explorer view. If you can see the project (*HockeyPool* in our case), one of your team members has already created it so you can

skip ahead to the “Exchanging source code with your team” section. If you don’t see the project, you will need to create it.

Create the project. Then, right-click the Project name and select *Team > Share Project...* . Select Jazz Source Control, click *Next*, and then select the component in your Repository Workspace (created in the “Creating your data stream” section above) and then select *Finish*.

To share the project with your team members, you must check-in to your repository workspace and then deliver to the stream. To check-in, right-click the Project name and select *Team > Check-in*. Now, to deliver the project to your team’s stream, right-click the Project name and select *Team > Deliver*.

Exchanging source code with your team

Editing code

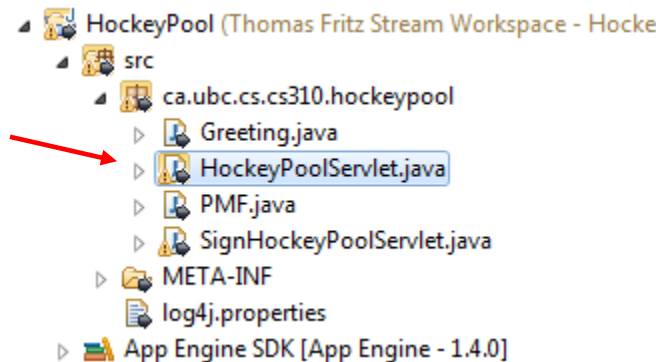
You should now **add classes or edit the code**.

Remember: every time you check-in a file it automatically gets backed up to your repository work space. You can therefore seamlessly work from your laptop, from home or any other machine!

Looking at your changes

Once you’ve modified some code, look for the yellow background on the file’s icon. The yellow background indicates that you have made some changes that have not yet been checked-in to your repository workspace (Figure 3).


Figure 3. Indication of change in the Package Explorer view.



To check-in these changes, right-click the file then select *Team... > Check-in*. Now your changes have been saved in your repository workspace.

Once you’re ready to share your changes with your team you need to *Deliver* the changes. Look for the Pending Changes view (it should be at the bottom of your screen).

Remember, if you can’t find a view just select it from *Window > Show View*.

The change symbol  identifies a change-set: that is, all the files you have modified since you last *delivered* your code to the stream. If you want to view these changes you can double-click on any java file and a Difference editor will appear, showing which lines were inserted and which were deleted.

Associating your change-set to a work item

In *Part I* of the tutorial we mentioned that every task you perform should correspond to a work item. This is especially important when editing code. So the changes you just performed to the code should correspond to a Work Item. For now, you should therefore **create a Task and name it *My first code edit***.

Normally, the work item should usually be created *before* you start editing code. This work item will be a *Task* when a new feature needs to be implemented, or a *Defect* if you are correcting a bug.

Now, in the Pending Changes view, **right-click on the change set and select *Associate Work Item***. Select the corresponding work item (you may have to type it in) and click *OK*. You should now see the description of the Work Item instead of *<Enter a Comment>* beside the change set.

If you specified that you are currently working on a Work Item – by changing its status to *Start Working* in the Work Item editor – then it will be automatically associated with your change set and you will not have to go through the preceding procedure.

Delivering a change set


To deliver your changes, right-click on the change set and select *Deliver and Resolve Work Item*.... This will bring-up a dialog in which you can add some comments to the Work Item. The changes will now be available to every other member of your team.

NOTE: You should only deliver changes to the team that compile and pass the tests. If the changes you deliver do not compile, it will slow down the whole team!

Failed delivery

The delivery operation may fail if one of your teammates delivered changes before you. This is because the system requires that you accept their changes before you can deliver yours. This ensures that you are all working on the same version of the project, and that potential conflicts are always resolved.

Accepting changes

To accept changes, click on the accept button . This could generate conflicts if the changes appear in the same file you modified. Don't worry, RTC will bring up a special editor to let you resolve these conflicts.

5 – Product and Sprint Backlog

Introduction

A software development process is usually divided in a number of iterations (also known as phases). The nature of these iterations will change based on the process model chosen by the team, but the *concept* of iteration remains. This concept is an integral part of RTC, and one way in which it is visible is through Iteration Plans.

In our case, we will use a Scrum process with one major release and two sprints (two iterations). If you are not familiar with Scrum, you should quickly read up on Scrum on the web to understand the basic concepts and in particular the next steps in our tutorial.

Creating a Product Backlog

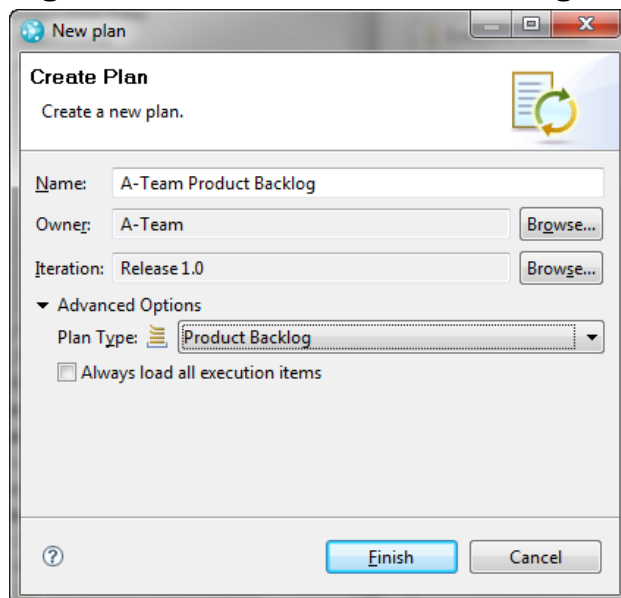
A product backlog contains all desired functionality for a particular release in the form of a prioritized list of product backlog items / user stories. It is used by an entire team.

Creating a product backlog can be done by any member of the team, but make sure only one of you is responsible for creating it, otherwise you will end-up with duplicates.

From the *Team Artifacts* view, go to *Plans > All Plans > Main Development > Release 1.0*, right-click and select *New > Plan...*

This will bring up a dialog (Figure 4). Name your iteration plan and **make sure the Owner is your team** and the Iteration is Release 1.0. Then expand the “Advanced Options”, select “Product Backlog” in the drop down menu and click “Finish”.

Figure 4. New Iteration Plan dialog.



Creating User Stories

You should now add the user stories for your project to the product backlog plan. When you open your Product Backlog Plan in the editor and go to the *Planned Items* tab, right-click on *Release 1.0* and you can add your user stories by selecting *Add Work Item > Story*. Fill in the summary of the user story in the text field.

Save the plan so that the user story is actually saved and open it in the editor by double-clicking the user story. Now add the relevant information, i.e. the story points (if you already decided on them), the priority and the acceptance criteria (under the acceptance test tab). Also make sure to file the user story against your team, i.e. select your team category in the *Filed Against* drop down menu (if the team category does not appear right away, click on *More...*, then you should be able to find it). You do not need to specify an owner yet.

Creating a Sprint Backlog

After the sprint planning meeting in which you decided on the user stories that you are going to include in the sprint, one of your teammates has to create the sprint backlog (basically an iteration plan). Again make sure **only one of you is responsible of creating it**. otherwise you will end-up with duplicates.

For creating the Sprint Backlog you have to make sure to create the sprint backlog in the appropriate sprint, i.e. go to *All Plans > Main Development > Release 1.0 > Sprint 1 (1.0)* for the first sprint, right-click it and create a new plan. Name your sprint backlog and don't forget to select your team as the owner and the sprint (Sprint 1 for the first sprint) in the Iteration, and select Iteration Plan in the Advanced Option section.

Each sprint backlog should have a short description for the main outcome of the sprint. **Click on the start editing link in the Overview tab and briefly describe what your team wants to do in the Sprint.** Click Save.

To move user stories from your product backlog into your sprint backlog, open your product backlog and drag and drop the user stories from the Release 1.0 section to the Sprint 1 section (in the *Planned Items* tab).

NOTE: if you do not see the Release 1.0 and Sprint 1 section in the *Planned Items* tab of the product backlog, select *Edit* under *View as* on the right side of the editor and then under *Grouping* select *Iteration*. Save this and the product backlog should now contain sections for the release and all sprints.

Dragging and dropping the items into the appropriate sprint will automatically associate them with the sprint and add them to your sprint backlog. So if you now open your sprint backlog again and go the *Planned Items* tab you should see the stories that you associated with the sprint.

Creating Sprint Tasks

Now that you have selected the user stories for your first sprint, you want to refine the user stories and create the actual tasks that you have to work on. So first open the sprint backlog. For now all user stories should be in the *Unassigned* section. Right-click on a user story and follow *Add Work Item > Task* to create a task. Fill in the summary field and press the save button to save the task. Open the task by double-clicking onto it and then fill in the other information. If the task is part of a user story, you should make this relation explicit by adding the user story as its parent (either by adding a parent in the task or by dragging and dropping the task onto the user story)

NOTE: Make sure to file your user stories and your tasks or defects against your team (i.e. choose the proper category in the *Filed Against* field). Otherwise, the story/task/defect will not show up in your product/sprint backlog.

When you decide to start working on a task or a user story, make sure to change the Owner to reflect you, so that everyone on the team can see that you are working on it.

At the end of an iteration, all the work items contained in the corresponding Iteration Plan must be resolved. If you have some work items left, you should either move them to the next iteration or resolve them, for example by indicating "Won't fix" in the appropriate field.

Since the Iteration Plan is such a great tool, you can make it part of your favourite pages. Just right-click on it in the Team Artifacts view and select *Add to Favorites...* It will now appear in your *Favorites* folder.

6 – Conclusion

Important concepts

Local Repository: The copy of the code that is on your hard drive; in the lab or at home.

Repository Workspace: Your own source-controlled backup, on the Jazz server.

Stream: The object through which teams exchange code.

Change set: A number of changes to various source files. A change set is a unit of changes that should be associated with a single work item.

Delivering: The act of moving code from your repository workspace to a stream.

Accepting: The act of moving code from a stream to your repository workspace.

Flow Target: The flow target of a repository workspace indicates the stream to which *deliver* and *accept* operations are performed.

What you should have learned

After completing this tutorial, you should have a basic understanding of the following important concepts related to Eclipse and RTC:

- The difference between your local workspace and the repository workspace;
- The concept of a *stream* to share code with other members of your team;
- How to create a stream and a repository workspace;
- The concept of a change set;
- How to associate a Work Item with a change set;
- How to *check-in* code to your repository workspace
- How to *accept* and *deliver* changes between you and your teammates;
- How to integrate your changes with your project teammates;
- That a software development process is a series of iterations;
- How to create a product and a sprint backlog;
- How to add user stories to a product backlog and to a sprint backlog;
- How to write the overview of a sprint backlog;
- How to add tasks to a sprint backlog and how to manage them.