serena.com

Best Practice for Parallel Development using Serena Dimensions

SERENA™

Table of Contents

Overview

This document is intended to help the reader to understand how to manage a number of typical Parallel Development scenarios using Streams in Serena Dimensions CM.

It should be kept in mind that Dimensions is an extremely flexible Software Configuration Management solution and the best practices offered here should be used as a guide when implementing your own processes.

It is assumed that the reader has a basic understanding of Dimensions and Software Configuration Management (SCM).

For detail step-by-step instructions on the topics discussed within this document, please refer to these Dimensions CM User Guides:

- *Dimensions User's Guide*
- *Dimensions for Visual Studio User's Guide*
- *Dimensions for Eclipse User's Guide*.

Types of Parallel Development

There are a great many valid approaches to parallel development, but for the purposes of this document we will be discussing four typical examples:

- Single Team – Single Development Stream

- Multiple Teams – Multiple Development Streams
    - IT Application Development
    - Independent Software Vendor (ISV)

For the purposes of this document parallel development means both a team of Developers working together on the same set of code for the same purpose and several teams of developers working on the same set of code for different purposes.

Parallel Development using Streams

What are Streams?

Streams are a type of container for development work similar to a project specifically designed to support the copy – modify – merge approach. This approach is typically used in agile or iterative style development, but is by no means exclusive to agile.

Streams have a number of properties not found in traditional Dimensions Projects:

- Streams enforce a single line of descent, preventing conflicts existing in the repository and enhancing the reliability of continuous builds.
- Developers resolve conflicts, build and test in their local development environment outside of Dimensions, improving reliability and removing the need for complex merges later on.
- Changes are committed atomically to the repository, meaning that Developers always get a consistent set of files when updating their work areas from the repository.
- Branching and Merging of Streams is simple and intuitive, allowing Developers to create Streams from within their favorite Integrated Development Environment (IDE) or user interface.

Streams are available in all of the interfaces in Dimensions CM and can be used with Dimensions Requests, Serena SBM Requests or with no Request management defined. Where Requests are referred to in this document either Dimensions Requests or Serena SBM Requests are applicable.

Traditional Dimensions Projects, that support a Lock – Modify – Unlock methodology, continue to be available in Dimensions CM. Projects, unlike Streams, do not automatically enforce a single line of descent. You can have parallel versions of the same code in a project because there can be more than one tip revision of the same item.

Streams and Projects are not mutually exclusive and can co-exist in the same Dimensions repository. This will enable an existing customer to gradually migrate from Projects to Streams, or to selectively introduce Streams to specific development teams while keeping other teams using Projects.

Please see the section "Who Should use Streams" later on in this document to help you decide which will best fit your development processes.

Copy - Modify - Merge

Streams implement a copy – modify – merge approach to version control. This means that Developers will:

> **Copy** – Use 'Update' work area command to copy files from the repository to their work area on disk.
> **Modify** – Make changes locally, then build and test locally.
> **Merge** – Use Dimensions to help merge any conflicting changes from the repository into their work area before 'Delivering' their changes to the repository.

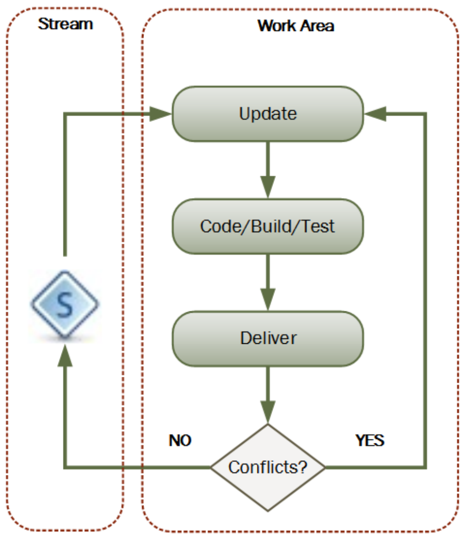


Figure 1: Copy - Modify - Merge

The copy-modify-merge model in practice runs extremely smoothly. Users can work in parallel, never waiting for one another. When they work on the same files, it turns out that most of their concurrent changes don't overlap at all; conflicts are infrequent. And the amount of time it takes to resolve conflicts when it does happen is far less than the time lost by a lock-modify-lock system.

Single Team – Single Development Stream

This is the simplest form of parallel development, and fairly typical of a modern Agile development scenario, but certainly not exclusive to Agile.

In this scenario a number of Developers will be working on the same set of code for a common purpose such as a sprint, a feature, a patch, or a release. Normally the developers will be making small incremental changes and want to regularly include changes from the repository. They will be delivering these changes to the repository using the copy – modify – merge approach. Each Developer is responsible for integrating, building and testing their changes locally before delivering them to the repository.

Update Work Area – Typically first thing in the morning and again before starting a new task a Developer will run 'Update'' work area to populate or refresh their work area. This can be run recursively for the whole Stream or for a sub folder, and will copy all files as writable. Conflicts can be auto-merged where possible or merged manually.

Code/Build/Test – The developer will make the necessary changes to complete their task, build and test locally to validate their work before deliver.

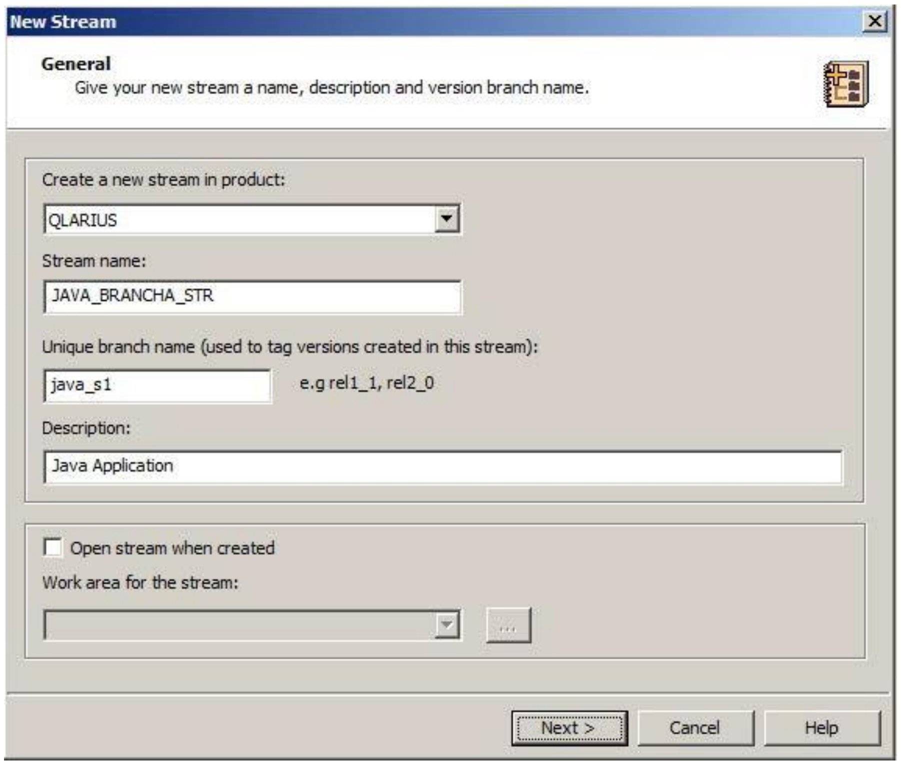
Deliver – When the Developer is satisfied that their task is complete, they will run the 'Deliver' command. This can be recursively on the whole work area, or on a sub folder. Dimensions will check, for the folders selected, if any of the files changed locally have also been changed in the Stream. If there are no conflicts, the local changes will be applied to the Stream. If any of the same files have changed, the Developer will be invited to run 'Update' work area command again to get the conflicting changes, resolve them locally then 'Deliver' them once they have built and tested locally.

This process is best explained using an example:

Rita the Team Leader is in charge of a development team who are about to start work on a new release of the application that they are responsible for. This is the only team who work on this application and there is no need for more than one development Stream.

Rita’s team has chosen to use either the Windows Explorer integration or the command line. Later on in this document we will cover using Eclipse and VS .Net.

In preparation for the first sprint Rita creates a new Stream, based on the final release Baseline for the previous release. She then instructs her team to use the new Stream.


New Stream

General
Give your new stream a name, description and version branch name.

Create a new stream in product: QLARIUS

Stream name: JAVA_BRANCHA_STR

Unique branch name (used to tag versions created in this stream): java_s1 e.g rel1_1, rel2_0

Description: Java Application

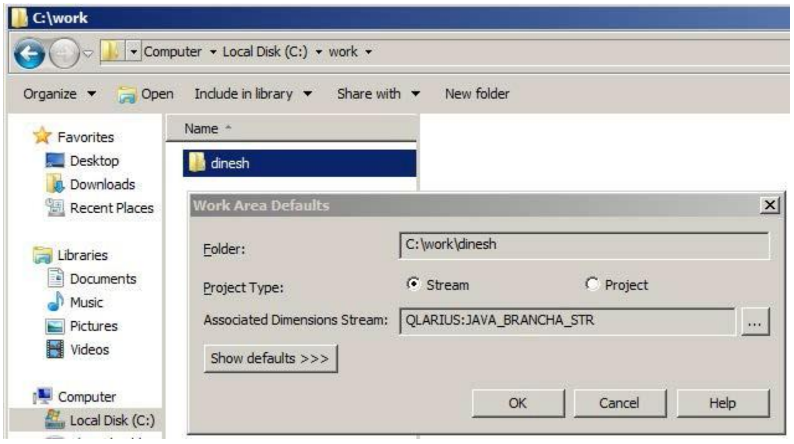
☐ Open stream when created

Work area for the stream:

Figure 2: Create Stream Wizard

Dinesh the Developer comes into work on the first working day of the new sprint keen to get started on the tasks allocated to him during the sprint planning session the day before. The tasks are represented by Requests so Dinesh only has to look at his In-box to see what he should be working on.

Having cleared out his work area from the last release, Dinesh uses his favorite UI, the Dimensions Windows Explorer integration to associate his work area with the new Stream and run 'Update' work area command from the root of the Stream.



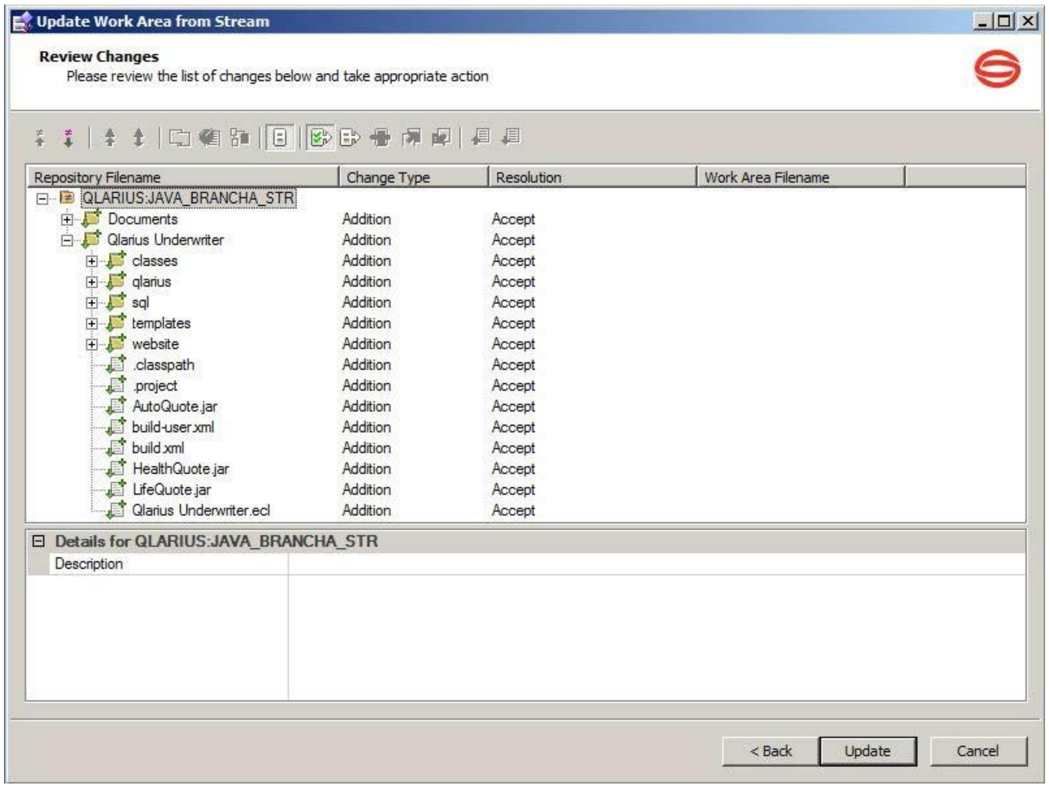
Work Area Defaults

Folder: C:\work\dinesh

Project Type: (● Stream) ○ Project

Associated Dimensions Stream: QLARIUS:JAVA_BRANCHA_STR

Figure 3: Associating a Work Area to Stream


Update Work Area from Stream
Review Changes
Please review the list of changes below and take appropriate action
< Back
Update
Cancel

Repository Filename	Change Type	Resolution	Work Area Filename
QLARIUS:JAVA_BRANCHA_STR			
Documents	Addition	Accept	
Qlarius Underwriter	Addition	Accept	
classes	Addition	Accept	
qlarius	Addition	Accept	
sql	Addition	Accept	
templates	Addition	Accept	
website	Addition	Accept	
.classpath	Addition	Accept	
.project	Addition	Accept	
AutoQuote.jar	Addition	Accept	
build-user.xml	Addition	Accept	
build.xml	Addition	Accept	
HealthQuote.jar	Addition	Accept	
LifeQuote.jar	Addition	Accept	
Qlarius Underwriter.ecl	Addition	Accept	

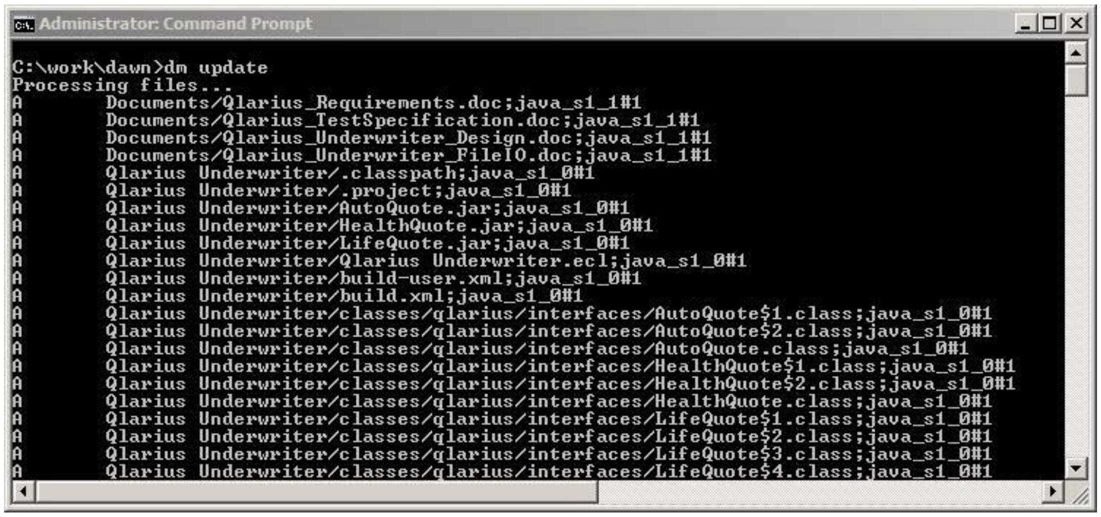
Details for QLARIUS:JAVA_BRANCHA_STR

Description

Figure 4: Update Work Area from Explorer indicating the files to be updated

Dinesh can now get on with his work.

At about the same time another member of the team, Dawn, is also starting to work in the new Stream. Dawn prefers to use the command line, so she cleans out her old work area, changes to the new Stream and runs 'dm update' from her command shell.



```
C:\work\dawn>dm update
Processing files...
A       Documents/Qlarius_Requirements.doc;java_s1_1#1
A       Documents/Qlarius_TestSpecification.doc;java_s1_1#1
A       Documents/Qlarius_Underwriter_Design.doc;java_s1_1#1
A       Documents/Qlarius_Underwriter_FileIO.doc;java_s1_1#1
A       Qlarius Underwriter/.classpath;java_s1_0#1
A       Qlarius Underwriter/.project;java_s1_0#1
A       Qlarius Underwriter/AutoQuote.jar;java_s1_0#1
A       Qlarius Underwriter/HealthQuote.jar;java_s1_0#1
A       Qlarius Underwriter/LifeQuote.jar;java_s1_0#1
A       Qlarius Underwriter/Qlarius Underwriter.ecl;java_s1_0#1
A       Qlarius Underwriter/build-user.xml;java_s1_0#1
A       Qlarius Underwriter/build.xml;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/AutoQuote$1.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/AutoQuote$2.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/AutoQuote.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/HealthQuote$1.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/HealthQuote$2.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/HealthQuote.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/LifeQuote$1.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/LifeQuote$2.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/LifeQuote$3.class;java_s1_0#1
A       Qlarius Underwriter/classes/qlarius/interfaces/LifeQuote$4.class;java_s1_0#1
```

Figure 5: Command line Update Work Area

The other members of the team follow the same process using their interface of choice.

The first change that Dawn makes is straightforward and she quickly has it coded, built and tested locally. So she runs the 'Deliver' command, specifying the Request she was working on, to put the changes back into Dimensions. There are no conflicts, so her changes are added to the Stream.

Sometime later Dinesh has also finished his changes and is ready to deliver them to Dimensions. He selects 'Deliver' at the root of the Stream (if Dinesh's changes had been restricted to just part of the folder structure, he could have selected just that folder to deliver from rather than using the root of the Stream).

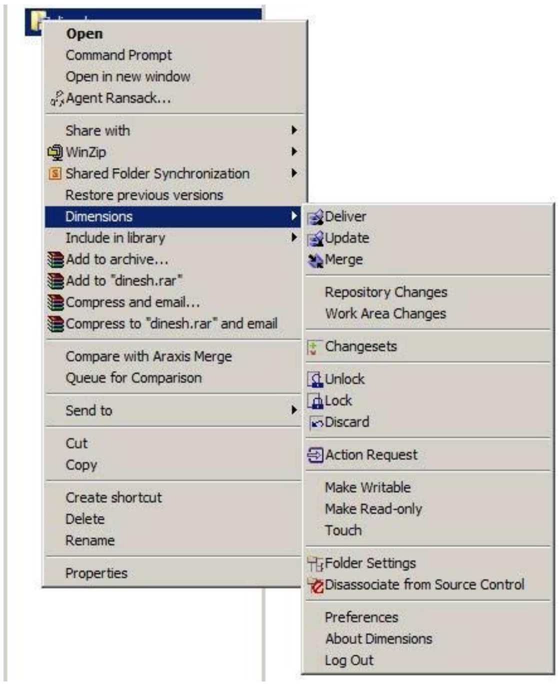
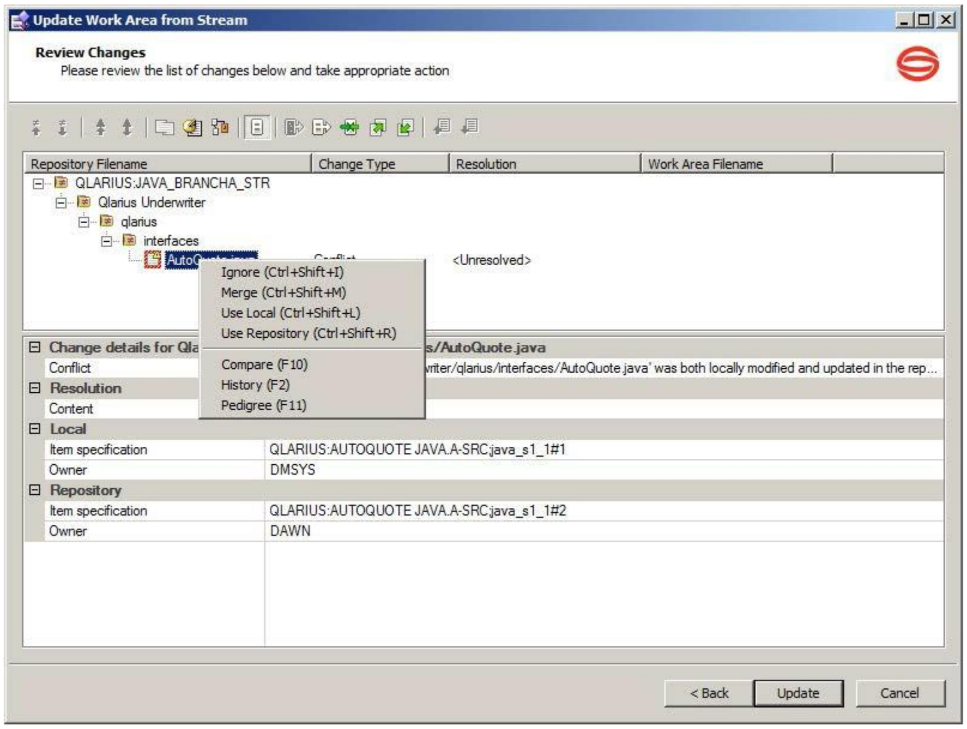


Figure 6: Deliver from the root of the Stream

Dinesh is warned that another Developer has already changed some of the files he has changed in the repository. As Dimensions enforces a single line of descent in Streams, Dinesh is forced to run 'Update' work area command to resolve these conflicts in his work area before he can deliver his changes. He runs the Sync tool to interactively merge the changes into his work area.



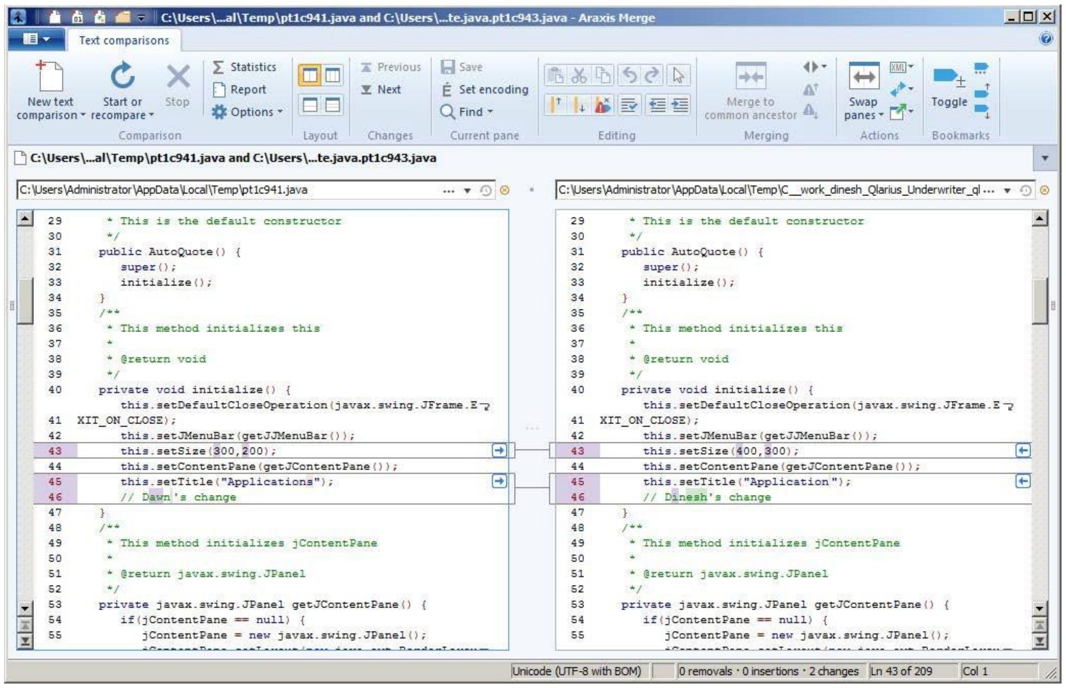
Review Changes
Please review the list of changes below and take appropriate action

Repository Filename	Change Type	Resolution	Work Area Filename
QLARIUS:JAVA_BRANCHA_STR			
Qlarius Underwriter			
qlarius			
interfaces			
AutoQ	Conflict	<Unresolved>	

Change details for Qla…s/AutoQuote.java	
Conflict	…riter/qlarius/interfaces/AutoQuote.java' was both locally modified and updated in the rep…
Resolution	
Content	
Local	
Item specification	QLARIUS:AUTOQUOTE JAVA.A-SRC;java_s1_1#1
Owner	DMSYS
Repository	
Item specification	QLARIUS:AUTOQUOTE JAVA.A-SRC;java_s1_1#2
Owner	DAWN

Figure 7: Sync Tool conflict resolution

Dimensions can automatically merge files, but where the same line has been changed in the both files, you will need to perform a manual merge, using the Serena supplied Araxis Merge tool or your preferred 3rd party tool.



C:\Users\...al\Temp\pt1c941.java and C:\Users\...te.java.pt1c943.java

C:\Users\Administrator\AppData\Local\Temp\pt1c941.java

```
    * This is the default constructor
    */
   public AutoQuote() {
      super();
      initialize();
   }
   /**
    * This method initializes this
    *
    * @return void
    */
   private void initialize() {
      this.setDefaultCloseOperation(javax.swing.JFrame.E
XIT_ON_CLOSE);
      this.setJMenuBar(getJJMenuBar());
      this.setSize(300,200);
      this.setContentPane(getJContentPane());
      this.setTitle("Applications");
      // Dawn's change
   }
   /**
    * This method initializes jContentPane
    *
    * @return javax.swing.JPanel
    */
   private javax.swing.JPanel getJContentPane() {
      if(jContentPane == null) {
         jContentPane = new javax.swing.JPanel();
```

C:\Users\Administrator\AppData\Local\Temp\C__work_dinesh_Qlarius_Underwriter_ql ...

```
    * This is the default constructor
    */
   public AutoQuote() {
      super();
      initialize();
   }
   /**
    * This method initializes this
    *
    * @return void
    */
   private void initialize() {
      this.setDefaultCloseOperation(javax.swing.JFrame.E
XIT_ON_CLOSE);
      this.setJMenuBar(getJJMenuBar());
      this.setSize(400,300);
      this.setContentPane(getJContentPane());
      this.setTitle("Application");
      // Dinesh's change
   }
   /**
    * This method initializes jContentPane
    *
    * @return javax.swing.JPanel
    */
   private javax.swing.JPanel getJContentPane() {
      if(jContentPane == null) {
         jContentPane = new javax.swing.JPanel();
```

Unicode (UTF-8 with BOM) | 0 removals · 0 insertions · 2 changes | Ln 43 of 209 | Col 1

Figure 8: File Merge tool

Dinesh will now build and test the merged changes locally before delivering them to the Stream.

This pattern will continue with each Developer regularly running 'Update' command to make sure they are working on the tip of the Stream and regularly delivering small incremental changes that they have built and tested locally.

Private Streams

In the normal course of events a development team working on the same codebase for the same purpose will share a Stream. This gives them immediate visibility of and access to each other's changes, improving collaboration and reducing the risk of a broken build. However there are some situations where it makes sense for an individual Developer to create a Stream to use in isolation from the main development Stream.

For example, a Developer is going to work on a high-risk change late on in the development cycle. They may well decide not to do this in the main development Stream in case it ends up breaking the build and not making the release. To do this they would create a new Stream based on the current main development Stream and assign their work area to the new Stream. From then on their changes would be delivered to the new Stream and would not be picked up in the main development Stream. Once the work was complete, it could be decided whether it was safe to merge these changes back into the main development Stream or not.

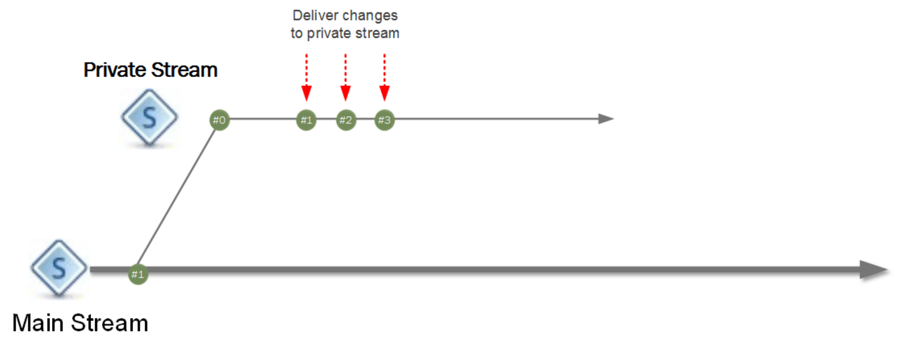


FIGURE 9: PRIVATE STREAM

NOTE: Having each Developer always create their own individual Stream is ***not*** recommended for two main reasons:

1. It is likely to result in a great number of Streams being created which can become a burden to administer.
2. It reduces the interaction and visibility within a team causing more complex and troublesome merges later on.

Multiple Teams – Multiple Development Streams

Life becomes more complex when multiple teams are involved working in parallel on the same code base, working towards the same or different purpose. There are numerous approaches to managing this type of environment, but for the purposes of this document we will look at two typical examples: IT Application Development and Independent Software Vendor (ISV).

IT Application Development Example

A formal protected mainline approach is often taken in Enterprise development environments providing critical applications to the business, where it is necessary to have multiple development projects running in parallel.

Here there is a mainline of code that is considered 'golden' and must never be broken. Changes are never made directly on the mainline, but a copy – modify – merge approach is taken. So a development team will branch from the mainline, make their changes and then merge their changes into the mainline. It is typical to have a role of Mainline Manager who acts as a gatekeeper for the mainline.

In this scenario a Stream is used to represent the mainline for an application or suite of applications. No work is done directly on this Stream; rather all work is done on Streams that have been branched from the mainline Stream.

In each stream a development team will typically follow a scenario similar to the single team – single stream example above.

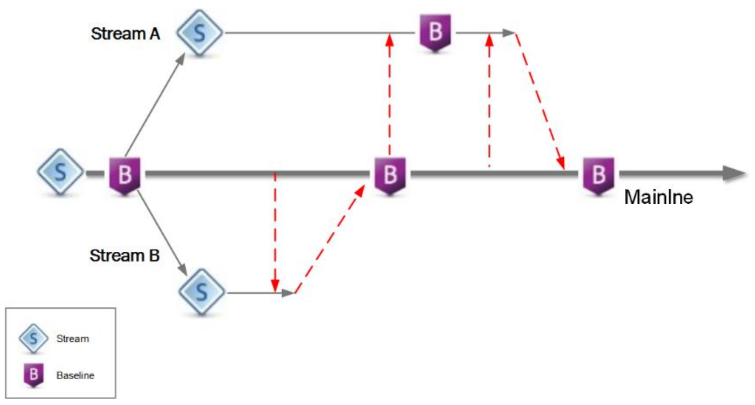


Figure 10: In House Development

Figure 10: In House Development

Doug the Development Manager is responsible for the development of his company's core business system. At any given time there are multiple projects being worked on of varying complexity.

To improve code stability and reduce defects resulting from merges, Doug has decided to define a Mainline from which each project will take a branch to work on.

In this example two projects are started at roughly the same time. Each Team Leader creates a new Stream based on the latest Baseline of the Mainline Stream. The two teams can then work independently of each other as described in the section "Single Team – Single Development Stream" above.

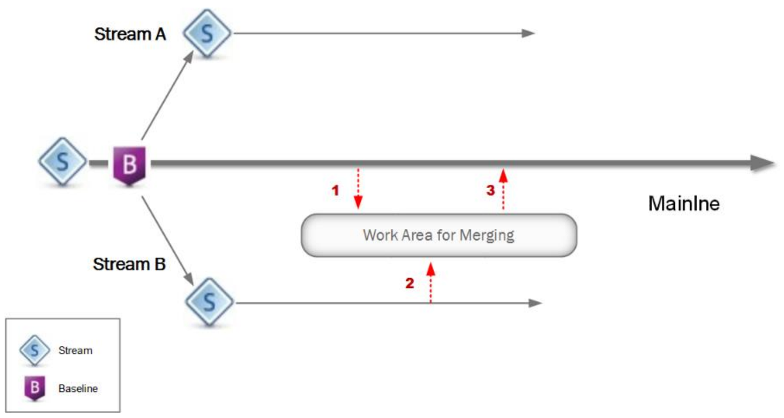
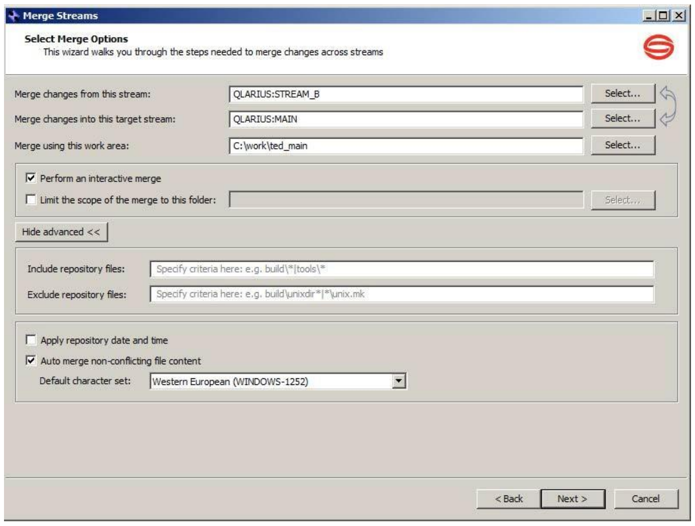


Figure 11: Merge Streams

Team B is doing a fairly small change and complete their work after a few days. To prepare for delivery of their changes to the mainline, Ted the Team Leader wants to get a feel for the complexity of changes involved. So he (1) runs 'Update' command from Mainline into a clean work area, then he (2) runs 'Merge' command to merge Stream B into the Mainline work area.


Merge Streams

Select Merge Options

This wizard walks you through the steps needed to merge changes across streams

Merge changes from this stream: QLARIUS:STREAM_B

Merge changes into this target stream: QLARIUS:MAIN

Merge using this work area: C:\work\ted_main

☑ Perform an interactive merge

☐ Limit the scope of the merge to this folder:

Include repository files: Specify criteria here: e.g. build*|tools*

Exclude repository files: Specify criteria here: e.g. build\unixdir*|*\unix.mk

☐ Apply repository date and time

☑ Auto merge non-conflicting file content

Default character set: Western European (WINDOWS-1252)

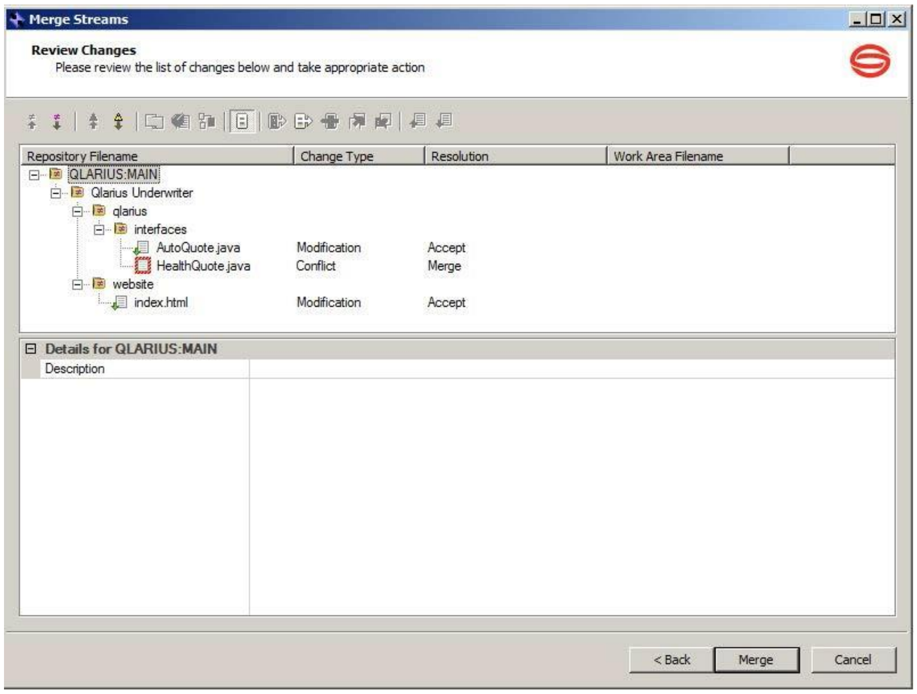
Figure 12: Merge Main Work Area from Branch Stream

This will update the Mainline work area with the latest changes from Stream B. The Merge tool will show that there are no conflicts to resolve because the mainline has not changed since Stream B was created. So Ted, who is the only person in the team who has permission to change the mainline, will be prompt to (3) 'Deliver' the changes to the Mainline Stream.

This will effectively make the tip of the Mainline Stream the same as the tip of Stream B. At this point a Baseline would normally be taken of the mainline Stream.

Sometime later Team A completes their work and want to deliver their changes to the Mainline Stream. To prepare for this delivery, Wendy the Team Leader wants to get a feel for the complexity of changes involved. So she runs 'Update' work area command from Stream A to make sure she has the tip of the Stream A, then 'Merge' from parent (Mainline) into her Stream A work area. The Merge tool will show that multiple changes have taken place in the mainline and that some of them conflict with changes made by Wendy's team.

So Wendy gets each of her developers to run 'Merge' command from parent (Mainline) into their work areas.


Merge Streams

Review Changes
Please review the list of changes below and take appropriate action

Repository Filename	Change Type	Resolution	Work Area Filename
QLARIUS:MAIN			
Qlarius Underwriter			
qlarius			
interfaces			
AutoQuote.java	Modification	Accept	
HealthQuote.java	Conflict	Merge	
website			
index.html	Modification	Accept	

Details for QLARIUS:MAIN
Description

Figure 13: Screenshot of resolving merge conflicts

Each Developer merges the changes that they are responsible for, build locally, test and Deliver the merged changes to their home Stream, Stream A. The whole team will work in this way until all merge conflicts have been resolved and Stream A is stable. At this point Wendy the Team Leader will lock the Stream to prevent any further changes, take a Baseline and run 'Update' command from Stream A to a clean work area. Then, having checked that nothing has changed in the Mainline, run 'Merge' to the Mainline Stream. This will effectively make the tip of the Mainline Stream the same as the tip of Stream A and the Mainline now contains working software that includes both Team A and Team B's changes.

This approach is entirely scalable to manage multiple development projects running in parallel without compromising the integrity of the mainline or imposing an uneconomical management overhead.

Releasing to Production could either be done from the mainline by creating a release Stream from the latest mainline Baseline in the same way as a development Stream; or releases could be made from each development Stream when they are complete before merging back into the mainline.

For a change-package oriented approach, when Requests are being used to record changes, Wendy could have selected the specific Requests she wanted to merge to the mainline Stream rather than taking the whole Stream. See 'Forward Fitting and Back Porting' below for more details.

Independent Software Vendor (ISV) example

In a commercial software house it is essential that patches to existing releases can be worked on without picking up changes from, or interrupting the development for the next release.

To support this, a Stream is created for all bug fixes and patches for a release while development for the next release continues in the main development Stream. Any changes made in the maintenance Stream can be merged into the main development Stream and into the current maintenance Streams. This maintenance branch can continue for as long as the release is supported.

Often development will start on the next release before the current release has shipped. In this case a Stream will be created, based on the release currently in development, to isolate this work. This stream will be merged back into the main development Stream once the release has shipped.

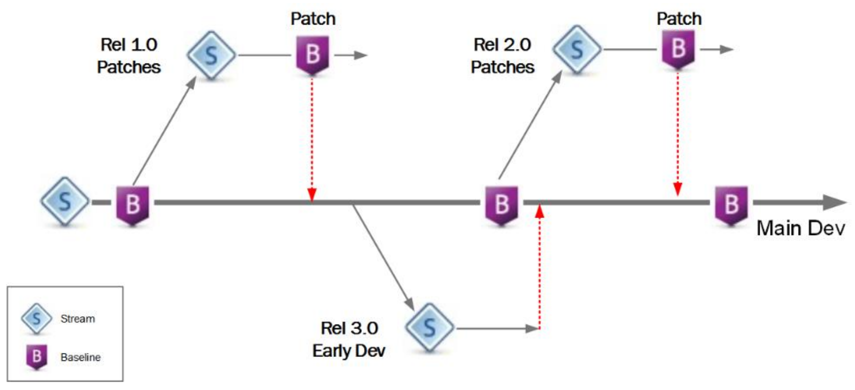


Figure 14: ISV Example

Debbie the development manager is responsible for the development of her company's latest software product. Release 1.0 has already shipped and is proving very successful and development for release 2.0 is well under way.

As expected there have been a number of defects reported against the first release that need to be fixed and released as a patch. Debbie does not want to distract the team working on Release 2.0, or accidentally pick up any of the work being done for the next release, so she asks the Maintenance Team leader, Josh, to create a new Stream from the Release 1.0 Baseline called 'Rel 1.0 Patches'.

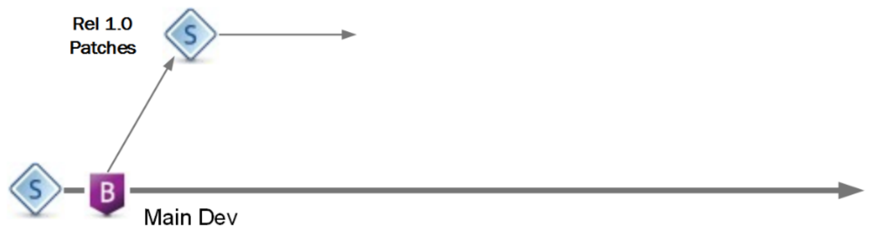


Figure 15: Rel 1.0 Patches

The first thing that Josh's team fixes is a memory leak for which they create a hot fix. This needs to be picked up in the development for the next release immediately, so Josh primes a patch request from the original defect and assigns it to the core development team.

In the core development team, Gill is given the task of forward fitting the memory leak defect to the main development Stream. He (1) runs 'Update' work area command from the main Stream to make sure he has the latest code. He (2) performs a 'Merge' get his changes from the patch request using the "Merge change request from another stream" option. This automatically selects all the changes made against the request and merges them into his work area. Although the Merge tool automatically provides Gill the option to Deliver his changes after the merge, he decides to first build and test locally, to make sure that the fix works correctly. He then (3) 'Delivers' the changes to the main Stream which includes the request fixing the memory leak.

The fix has now been included in the development for the next release and there is an audit trail back to the original defects. He continues with additional changes to the Rel_1_Patch Stream without hindering the Main development stream.

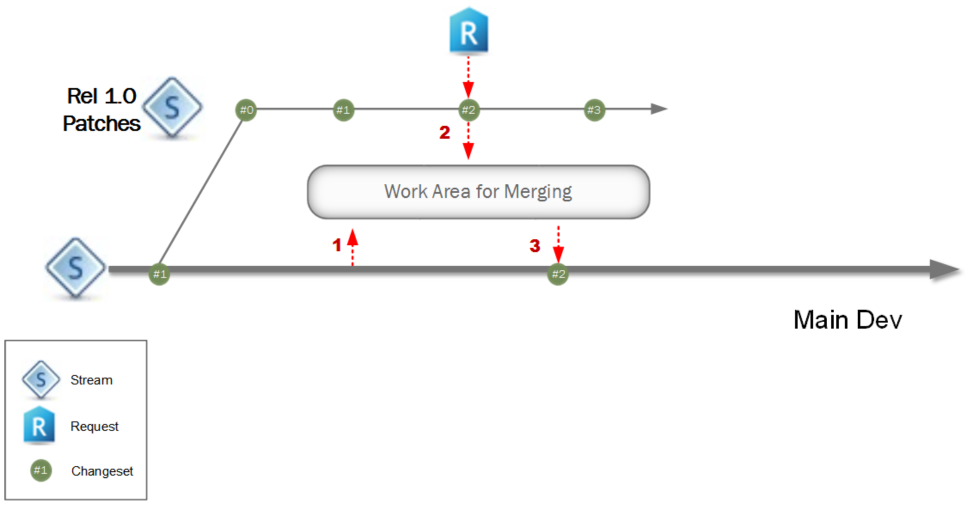


Figure 16: Merge Request into main Stream

There are some major architectural changes planned for Release 3.0 that will take longer than the release cycle to complete. So a new Stream (Rel 3.0 Early Dev) is created from the current development Stream so that this work can start without affecting the work for the current release.

Once Release 2.0 is out of the door, and the patches Stream has been created, the changes in 'Rel 3.0 Early Dev' can be merged back into the main development Stream. This would be achieved by one or more developers creating a clean work area from the main development Stream and then running 'Update' work area command from the 'Rel 3.0 Early Dev' Stream, resolving conflicts, build and test locally and 'Merge' the results to the main development Stream.

The pattern repeats for each new main and patch release. This approach is scalable to multiple parallel Streams of development being managed at the same time.

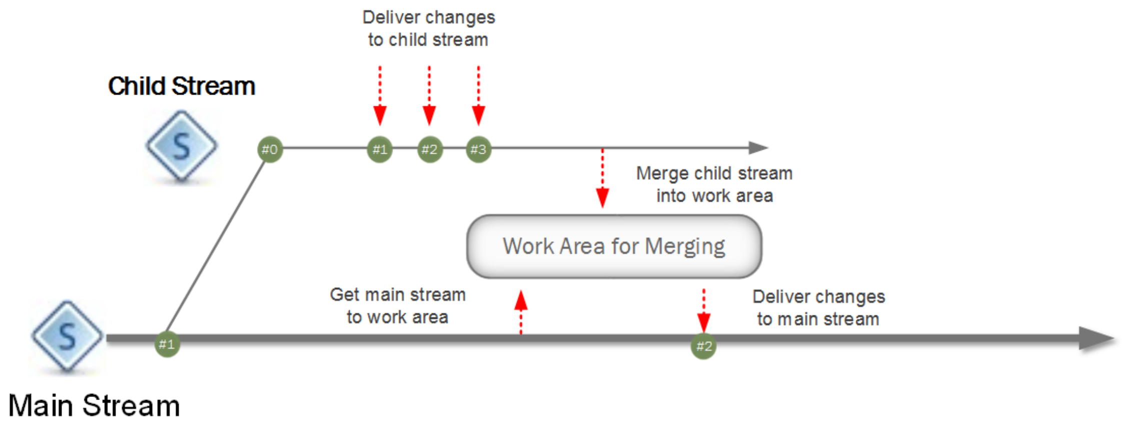

Deliver changes to child stream
Child Stream
#0
#1
#2
#3
Merge child stream into work area
Work Area for Merging
Get main stream to work area
Deliver changes to main stream
#1
#2
Main Stream

Figure 17: General Process of Merging Streams

Forward Fitting and Back Porting

It is normal that a fix made in one development Stream will also be needed in other parallel Streams and/or the mainline, if you are maintaining a mainline. This is often referred to as forward fitting or back porting. The best way to achieve this is to use a Request to capture the changes for each fix; this can then be used to apply the change to other Streams via the developer's work area.

Note: Merging changes from requests owned by another stream is only available in the desktop client and window explorer client.

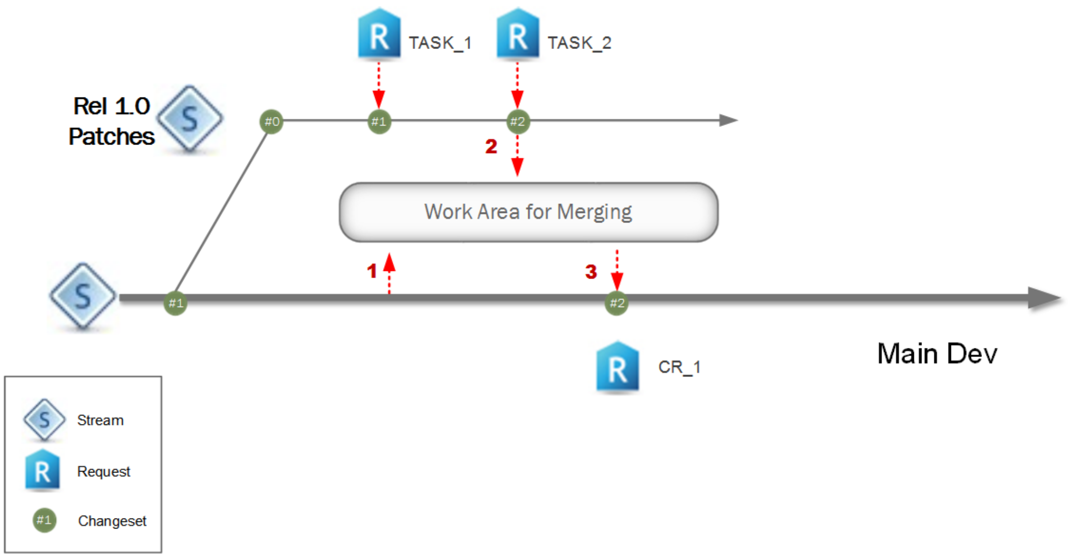


Figure 18: Forward Fit

In the diagram above the developer completes the changes in the 'Rel 1.0 Patches' Stream and relates the files to their assigned request (TASK_2). A change request (CR_1) is primed from TASK_2 and assigned to a lead responsible for the 'Main' stream.

The lead (1) runs 'Update' work area command from the 'Main' Stream to make sure they have the latest code. Then (2) performs a 'Merge' get his changes from the task (TASK_2) using the "Merge change request from another stream" option. This automatically selects all the changes made against the task and merges them into the work area. T he Merge tool automatically provides the option to deliver the changes after the merge, the lead could decide to first build and test locally, to make sure that the fix works correctly, but chooses to deliver into the 'Main' stream.

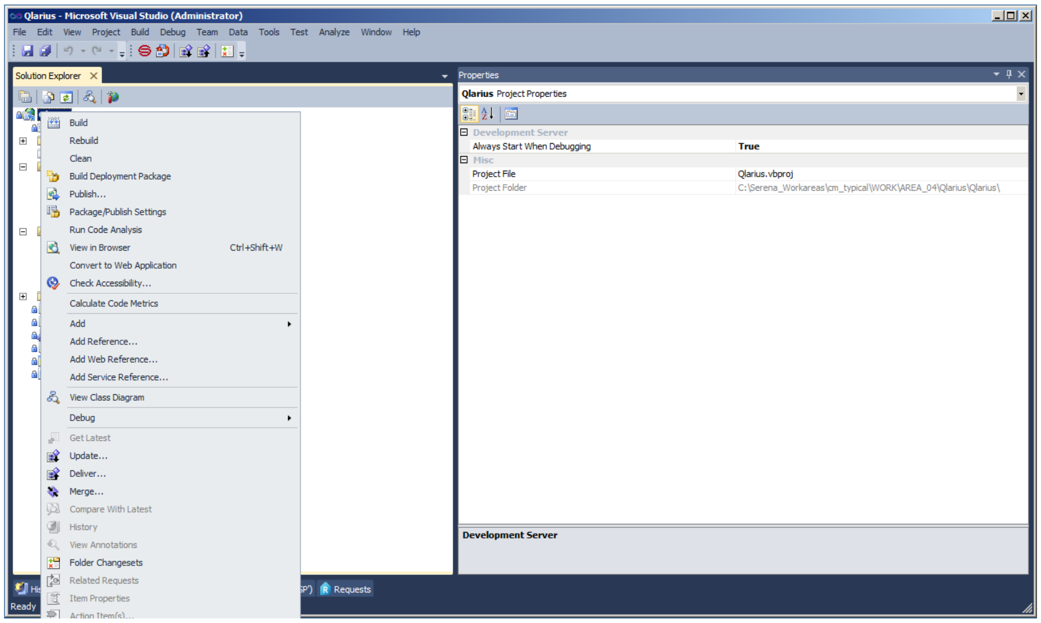
The fix has now been included in the 'Main' stream for the next release and there is an audit trail back to the original task because of the relationship between TASK_2 and CR_1.

Where a number of fixes are being merged, it is sensible to have a parent change that all the individual changes are related to. This parent change can then be used for the operations above rather than having to select multiple Tasks and run the risk of missing one.

Streams in Visual Studio

Streams are fully supported in the Visual Studio integration and the scenarios described above could have been performed using this integration. For more information, see the *Dimensions for Visual Studio User's Guide*.

Update and Deliver commands can be performed on a Visual Studio Solution or Project. The only significant difference being that when you first add a Visual Studio Solution or Project to Dimensions you use the 'Add to source control' command and when you want to bring a Visual Studio Solution or Project down to disk you use the 'Open from Source Control' command.

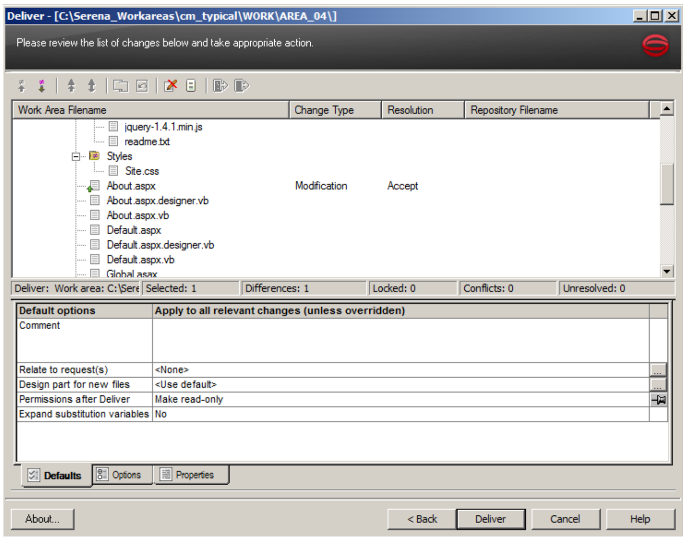


Qlarius Project Properties

Development Server	
Always Start When Debugging	True
Misc	
Project File	Qlarius.vbproj
Project Folder	C:\Serena_Workareas\cm_typical\WORK\AREA_04\Qlarius\Qlarius\

Development Server

Figure 19: Visual Studio Integration



Deliver - [C:\Serena_Workareas\cm_typical\WORK\AREA_04\]

Please review the list of changes below and take appropriate action.

Work Area Filename	Change Type	Resolution	Repository Filename
jquery-1.4.1.min.js			
readme.txt			
Styles			
Site.css			
About.aspx	Modification	Accept	
About.aspx.designer.vb			
About.aspx.vb			
Default.aspx			
Default.aspx.designer.vb			
Default.aspx.vb			
Global.asax			

Deliver: Work area: C:\Sere Selected: 1 Differences: 1 Locked: 0 Conflicts: 0 Unresolved: 0

Default options	Apply to all relevant changes (unless overridden)
Comment	
Relate to request(s)	<None>
Design part for new files	<Use default>
Permissions after Deliver	Make read-only
Expand substitution variables	No

Figure 20: Deliver from Visual Studio

Streams in Eclipse

Streams are fully supported in the Eclipse Integration, although they are accessed via the native Eclipse Synchronize tool rather than via the Dimensions Sync tool. The scenarios described above could have been performed using this integration. . For more information, see the *Dimensions for Eclipse User's Guide*.

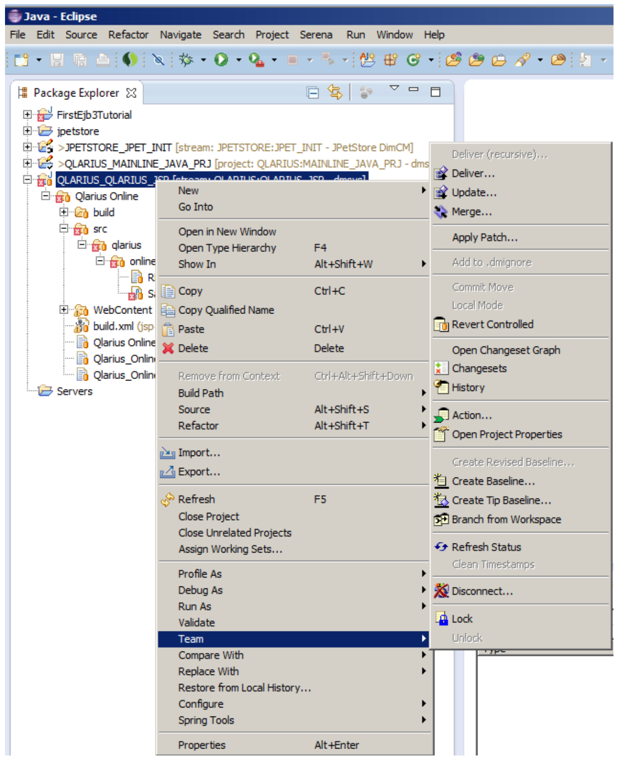


Figure 21: Eclipse Integration

Who should use Streams

When should a customer use a Stream rather than a Project?

Generally speaking Streams are aimed at development teams who are using a copy – modify – merge approach. Traditional Dimensions Projects are better suited to a lock – modify – unlock approach and non-software uses such as documentation, hardware assets etc.

If any of the following statements apply to your development project, you should consider using Streams:

- We are adopting or are already doing agile software development.
- We typically have a team of developers working in parallel on the same code for the same release.
- We want to manage several parallel streams of development with a protected mainline.
- We want to enable Developers to easily branch and merge projects without needing an administrator.
- We want to enforce a single line of descent to prevent conflicts.
- We want to make Developers responsible for the impact of their changes on the rest of the application.
- We want to use a 'copy - modify - merge' approach rather than taking an exclusive lock.

If any of the following statements apply to your development project, you should use traditional Dimensions Projects:

- We primarily manage non-software assets such as documentation and hardware.
- We are in a regulated environment where it is necessary to pre-authorize each change before it made.
- We follow a traditional waterfall development methodology.
- We use an SCC-based IDE for our development

Streams Best Practices

Below are a list of tips presented at Serena xChange by Richard Minchin, Director of Software Development for the Dimensions CM R&D team.

1. For large applications with a number of components, consider the following:
   - Use design parts to model the components
   - Put all code into a single stream
   - Place each component in their own folder structure
   - Do not try to use sub-projects for component based development

2. Organize your source code structure in a way that developers can download distinct chunks of it (components)
   - Work on those components and deliver them back
   - Avoids the need for a developer to have the WHOLE stream in thier work area to do their work.
   - Improves UPDATE and DELIVER performance

3. Recommended approach for merging streams:
   - Create a new work area and update from home stream
   - Lock the home stream during merge
   - Update folder or complete stream from the foreign stream
     - Files can be marked as merged permanently using MI /SELF
   - Build, Fix, Build, Test
   - Deliver using Ignore filters to skip build artifacts
   - Merge next folder (repeat above)
   - Run stream compare to see and check results of merge

4. Do relate requests to streams
5. Lock the home stream during your merge
6. Use Stream | Compare to identify folders you need to merge
7. Consider 'granting' privileges rather than allocating based on role
8. Utilize "Update files from project/stream" and "Deliver files to project/stream" privileges

FAQ

Q1: Should each Developer have a private Stream to work in?

A: No, as a general rule a team of developers working towards the same goal should share a Stream. An example of this would be a sprint team working on a specific feature, or a maintenance team working on a patch release. It would be good practice for a Developer to only create a private Stream when absolutely necessary, for example if they have to work on a risky change without destabilizing the build before merging this back into the main development Stream.

Q2: Can I mix and match Streams and Projects within my organization?

A: Yes, it is perfectly reasonable to have some teams using Streams and others using traditional Projects.

Q3: Can I create a Stream from a Project?

A: No, you can only create a Stream from another Stream or a Baseline to avoid the potential of conflicts being introduced into the Stream.

Q4: Can I merge a Project and a Stream?

A: You cannot merge a Project into a Stream, but you can merge a Stream into a Project.

Q5: As an existing Dimensions user, how do I convert my traditional Projects into Streams?

A: You will need to create a Release or Project Baseline from your current development Project and then create a new Stream from this Baseline. This new Stream will now represent your 'mainline' Stream from which you can create a branch Stream for development work. See the section 'Getting Started with Streams' in this document for more detail.

Q6: What happens to Developers work areas when they have converted to using Streams?

A: Following the conversion from traditional Projects, each Developer will need to run the 'Update' work area command on their work area before they start work.

Q7: Can I use the traditional check-in/check-out/Get/Update commands in a Stream?

A: No, Streams use optimistic locking and the only way to populate your work area is using the 'Update' work area command and the only way to add files to the Stream is using the 'Deliver' command.

ABOUT SERENA

Serena Software provides Orchestrated application development and release management solutions to the Global 2000. Our 2,500 active enterprise customers, including a majority of the Fortune 100, have made Serena the largest independent Application Lifecycle Management (ALM) vendor and the only one that orchestrates DevOps, the processes that bring together application development and operations.

Headquartered in Silicon Valley, Serena serves enterprise customers across the globe. Serena is a portfolio company of HGGC, a leading middle market private equity firm. For more information on Serena, visit http://www.serena.com.

CONTACT

Learn more about the enterprise-wide power of Serena products by visiting www.serena.com or contacting one of our sales representatives in your area.

Serena Worldwide Headquarters Serena Software, Inc. Corporate Offices 1850 Gateway Drive. 4th Floor San Mateo, California 94404 United States	Serena European Headquarters Serena Software Europe Ltd. Hertfordshire Abbey View Everard Close St. Albans Hertfordshire AL 1 2PS United Kingdom	Serena Asia Pacific Headquarters Serena Software Pte Ltd. 51 Bras Basah Road #06-08 Manulife Centre Singapore 189554
800.457.3736 T 650.481.3400 T 650.481.3700 F info@serena.com	+44(0) 1727.812.812 T +44(0) 1727.869.804 F ukinfo@serena.com	+65 6834.9831 T +65 6836.3119 F apinfo@serena.com