

**Easy! Software LLC**

---

# **Easy!Flow SCMClient for Eclipse User Guide**

Easy! Software LLC  
General Business Information

This document contains information, which is under protection of copyright. Any distribution, revision and publication of this document are prohibited. The usage of this document must be under the authorization from Easy! Software LLC,

## Table of Contents

<b>1. Getting Started</b> .....	3
1.1. What is Easy!Flow SCMClient.....	3
1.2. Why Easy!Flow SCMClient.....	3
1.3. Edit Config File.....	4
1.4. A Simple Example.....	4
<b>2. Working with SCMClient</b> .....	8
2.1. Login to SCMClient.....	8
2.2. Open Easy!Flow Site.....	8
2.3. Create SVN Product.....	9
2.4. Create Branch.....	10
2.5. Get Local Copy and Commit.....	11
2.6. Merge and Conflict Resolution.....	13
2.7. Backout Merge.....	15
2.8. Tagging a Baseline.....	16
<b>3. FAQ</b> .....	17

## 1. Getting Started

### 1.1. What is Easy!Flow SCMClient

Easy!Flow SCMClient is an application module that provides an effective and easy-to-use change management process. It helps user to interface with Subversion (SVN) through a pre-defined, configurable, and well-managed process. With the Easy!Flow SCMClient, users no longer need to worry about SVN repository schema, change control rules, how to find the revision to merge or back out and so on. It does these all for the user automatically.

Easy!Flow offers two types of SCMClient: (1) Stand-alone SCMClient, (2) SCMClient for Eclipse. The latter allows its users to access Easy!Flow and perform SVN related operations directly from Eclipse IDE.

### 1.2. Why Easy!Flow SCMClient

The following points list the benefits of Easy!Flow SCMClient over other SVN clients and raw SVN commands.

1. Easy to use -- Unlike other SVN clients, which just present all SVN commands to users, Easy!Flow SCMClient provides a simple and process oriented interface to its users. Users will no longer need to take care of the repository schema, branching & tagging strategy, revisions of individual change set, and so on, they can accomplish all these in a single window.
2. Server-side trigger -- Easy!Flow SCMClient provides server side trigger to prevent users from process violation, and it guides its users through their work. No more frustrations due to too many trigger rejections.
3. Organize your repository automatically -- Easy!Flow SCMClient wrapped the unified repository schema suitable for change management and tracking. The repository schema is automatically organized and protected by Easy!Flow SCMClient and server-side trigger.
4. Seamlessly integrated with Easy!Flow Tracking Ticket System and Easy!Review -- Easy!Flow SCMClient maintains data consistency between tracking records in Easy!Flow Tracking Ticket System and actual changes in SVN repository. Easy!Review automatically generates review page for changed files.

### 1.3. Edit Config File

It is simple to configure SCMClient. Select **SCMClient > Edit Config File**, open the configuration file with your favorite text editor, and set proper URL address for 'easyflow\_url' and 'repos\_url'. ('instance' does not need to be changed, unless you have multiple Easy!Flow instances)

See below contents as an example:

```
# Remote Easy!Flow Tracking Ticket System.
easyflow_url=http://10.0.0.102/EasyFlow/

# The DB instance used in Easy!Flow Tracking Ticket System.
# Separate instances with ';'
# For example: instance=default;instance1
instance=default

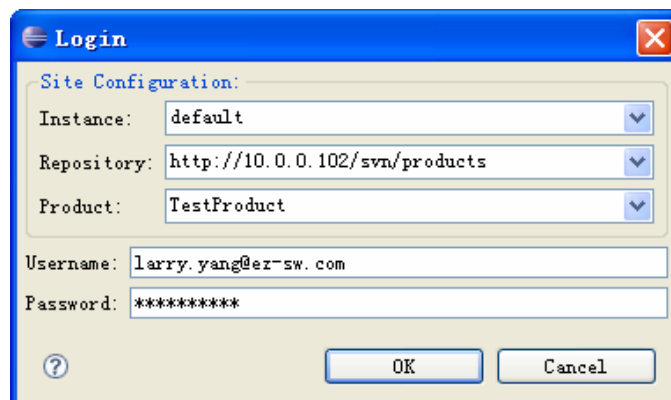
# Remote Repository.
# Separate repositories for different instances with '|'.
# Separate repositories in the same instance with ';'.
# For example: repos_url=;|;
repos_url=http://10.0.0.102/svn/products/
```

### 1.4. A Simple Example

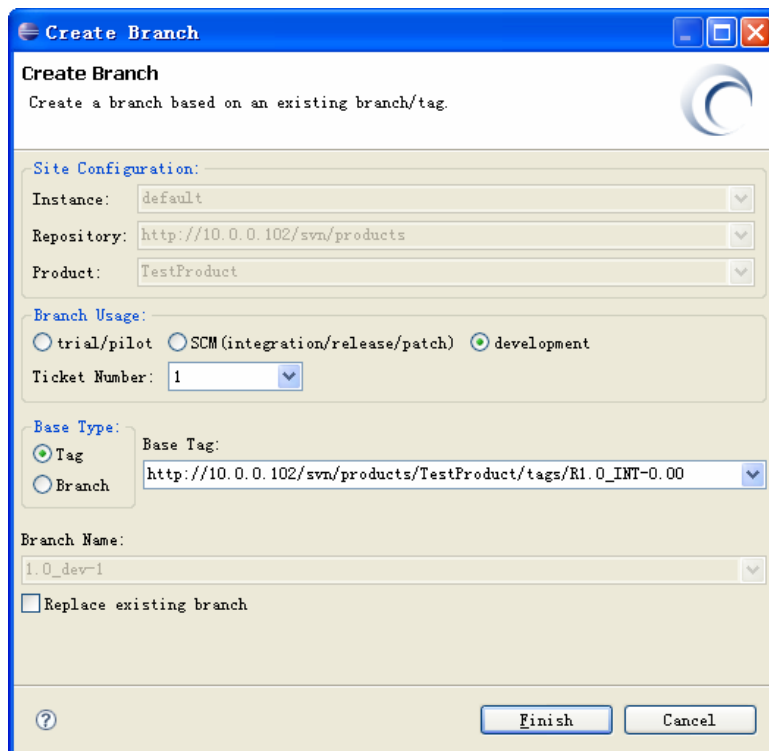
The following is a simplified example of using SCMClient for Eclipse.

Assuming Larry is working on tracking ticket (say #1) and needs to make changes on a few files, the following example illustrates how he uses Easy!Flow SCMClient to complete the change and integration cycle.

Step 1: Larry logs into SCMClient as shown below:

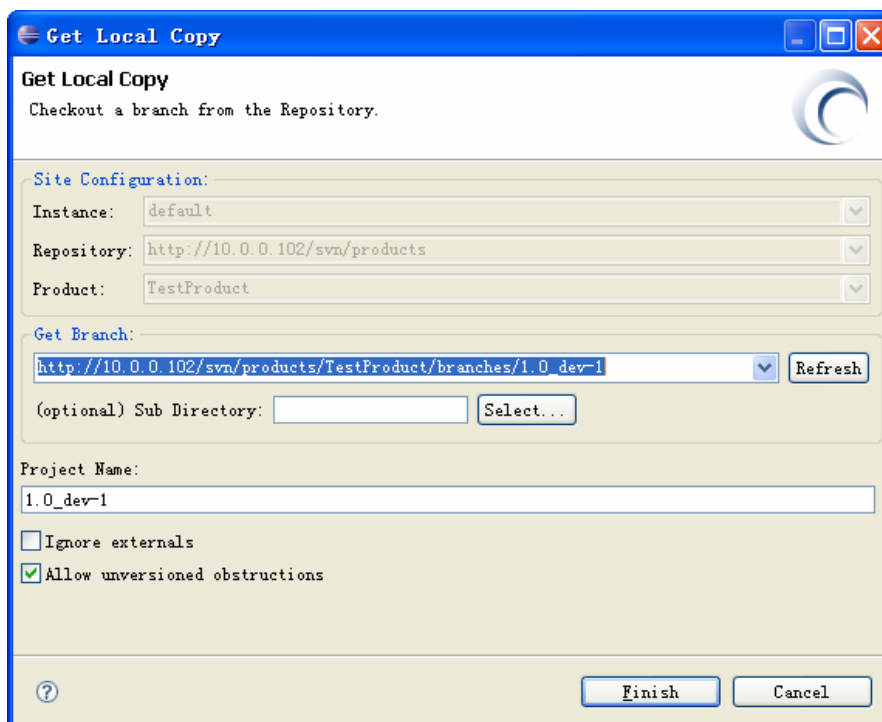


Step 2: Larry creates a branch to hold his own changes as shown below. The branch is dedicated to hold his changes without impacting others.



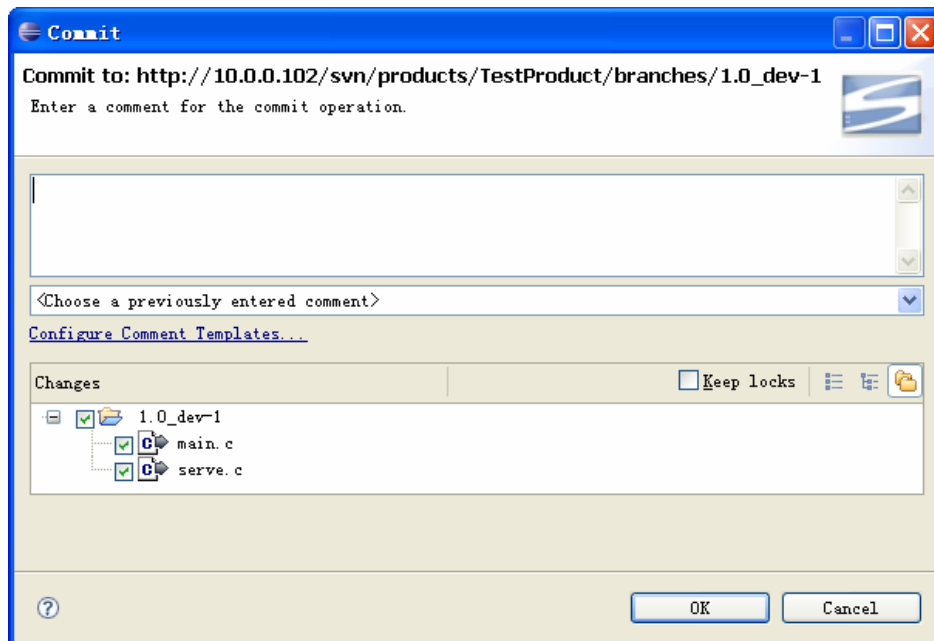
**Note:** The SCMClient will automatically call up SVN console for displaying operation message. The console is located at the bottom of the Eclipse UI,

Step 3: Larry checks out a local copy of files from his branch:

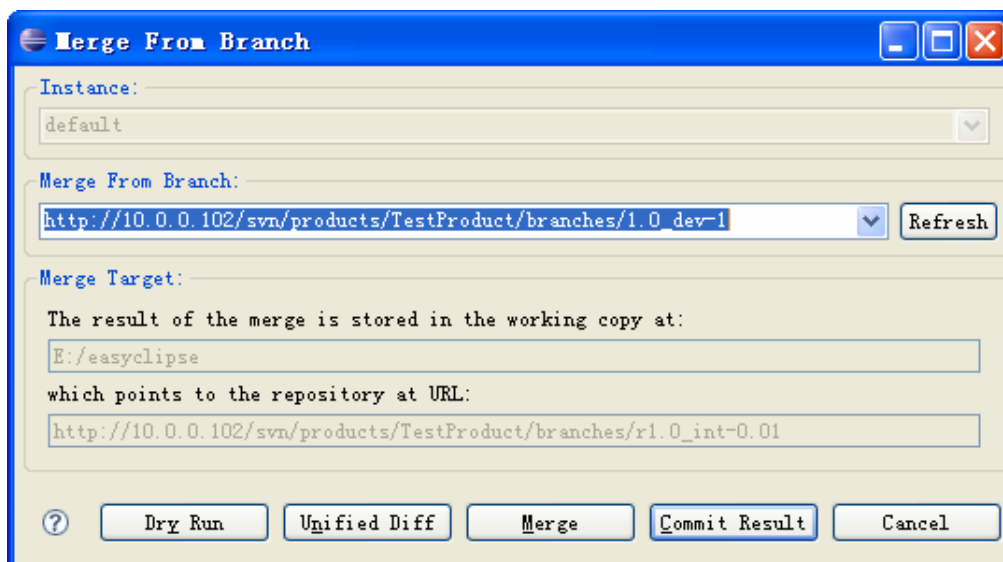


## Easy!Flow User Guide Documentation

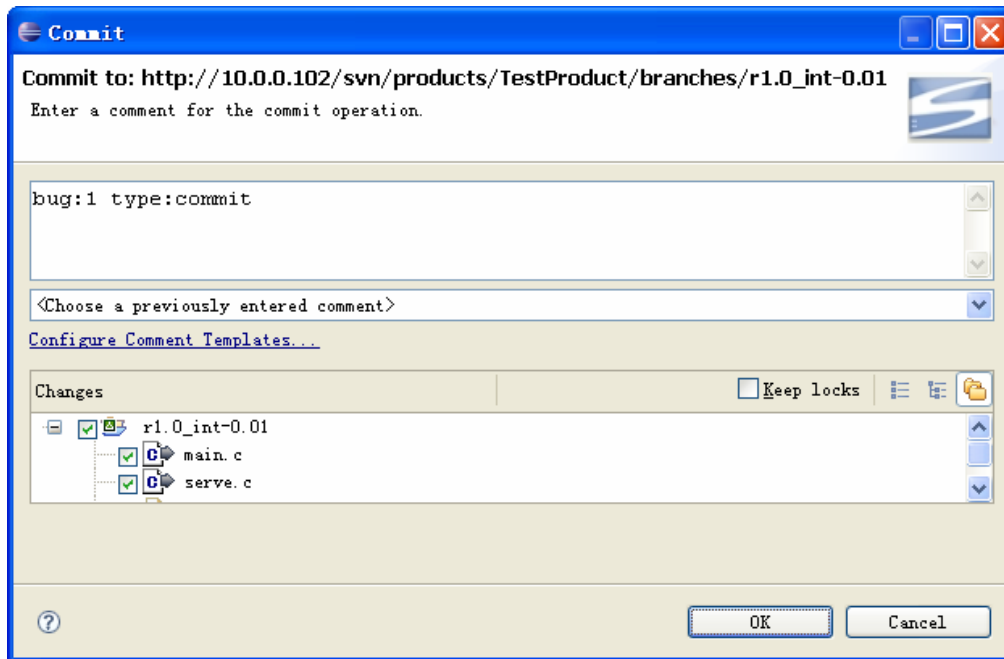
Step 4: Larry works on his changes (in the local copy), saves the changes then commits the changed files to his branch in the repository, as shown below.



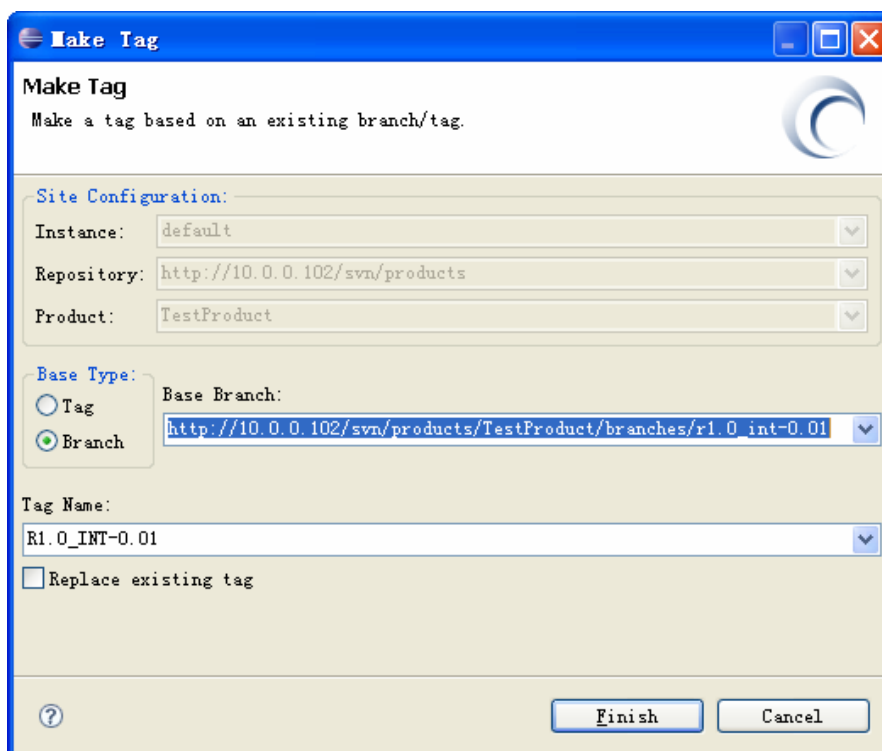
Step 5: Larry merges his changes from his development branch to (the local copy of) the integration branch (if the local copy of the integration branch was not created before the merge, he will have to create one).



Step 6: If there is no conflict encountered during the merge, Larry closes his changes by committing the local copy of the integration branch to the integration branch in the repository.



Step 7: After the integration is done, the system administrator makes a tag on the integration branch and this tag will be used as the base tag for the next change cycle.



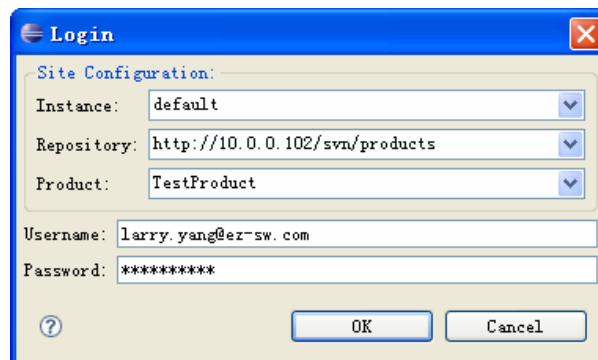
As described above, construction of product source code structure, trigger checkup, tracking record update, SVN commands generation and execution will be performed by the Easy!Flow SCMClient automatically. This frees users from studying how SVN works, struggling through raw SVN commands, and worrying about business rules. So they can focus on what they are supposed to do: create solid and defect-free product.

## 2. Working with SCMClient

### 2.1. Login to SCMClient

#### UI Path: SCMClient > Log In

The login user account is the same as that of Easy!Flow. All SVN operations in SCMClient will be performed only if the login user is authorized to do so. In the login window, you need to choose an Easy!Flow Instance that your product is associated with, a Repository -- the remote repository where your product is hosted, and the Product you'd like to access in that repository.



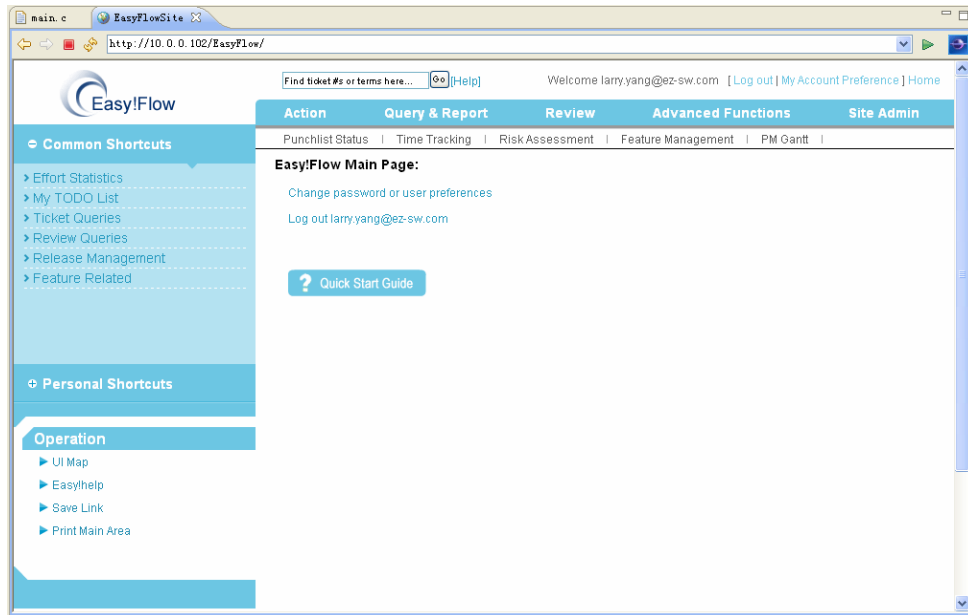
**Note:** The available Instances and SVN repositories are defined in SCMClient Config File, which you can edit directly from SCMClient>Edit Config File.

### 2.2. Open Easy!Flow Site

#### UI Path: SCMClient > Open Easy!Flow Site

Users can switch to the Easy!Flow UI via SCMClient>Open Easy!Flow Site. The SCMClient will automatically perform login for a user.

# Easy!Flow User Guide Documentation

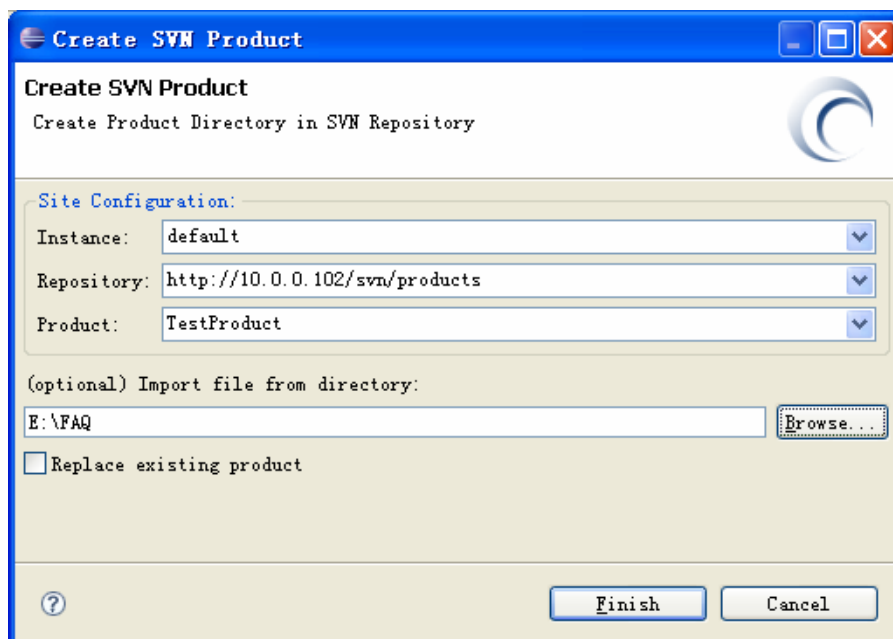


## 2.3. Create SVN Product

### UI Path: SCMClient > Create SVN Product

After a new product is created in Easy!Flow Tracking Ticket System, an administrative user should create the product in SVN repository as well.

If you have initial files to import to the new product, click "Browse..." button and select the directory containing the initial files.



## 2.4. Create Branch

### UI Path: SCMClient > Create Branch

Branching is the main strategy in modern change management system to realize simultaneous multi-users change coordination. Easy!Flow SCMClient has three types of branches: trial, development and official SCM branches. The trial branch is used for an individual to access and change code based on a specific tagged baseline, and the changes made won't be incorporated into official code. Development branch is used for official change requested via a tracking ticket; development branch will be merged to the Official SCM branch for change integration. Official SCM branch is used for integration of file changes from different development branches to form a new release or baseline.

The naming convention of the three types of branches is as follows. Easy!Flow SCMClient will automatically form branch names (partially or fully) with the information provided by a user.

Branch type	Naming convention	Example
Trial	private_tmp-<free text> or free format	private_tmp-test-by-larry
Development	<product version>_dev-<tracking ticket number>	1.0_dev-1
Official SCM Branches	The same as the milestones defined for the product in Easy!Flow Tracking Ticket System	r1.0_int-0.01

Generally, a user is working on a branch that is based on another branch or tag. Easy!Flow SCMClient will query available branches or tags pertaining to the selected product, such that a user only needs to select one from the "Base tag/branch" dropdown list.

**Create Branch**

Create a branch based on an existing branch/tag.

**Site Configuration:**

Instance: default

Repository: http://10.0.0.102/svn/products

Product: TestProduct

**Branch Usage:**

trial/pilot  SCM (integration/release/patch)  development

Ticket Number: 1

**Base Type:**

Tag  Branch

Base Tag: http://10.0.0.102/svn/products/TestProduct/tags/R1.0\_INT-0.00

**Branch Name:**

1.0\_dev-1

Replace existing branch

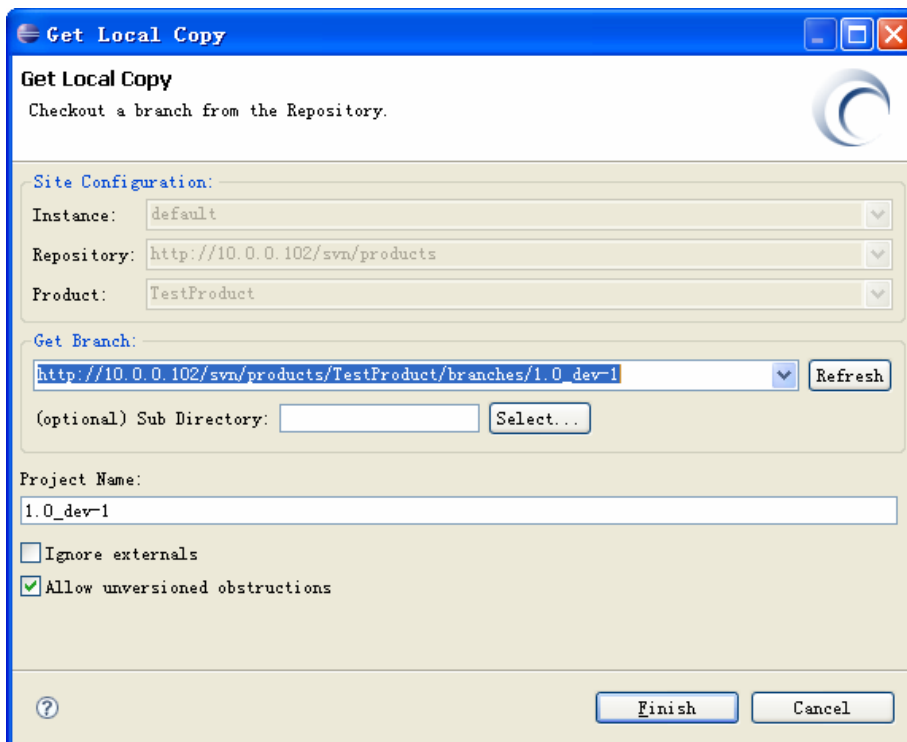
Finish Cancel

If a branch exists already, user can check the "Replace existing branch/tag" option box to make a replacement.

## 2.5. Get Local Copy and Commit

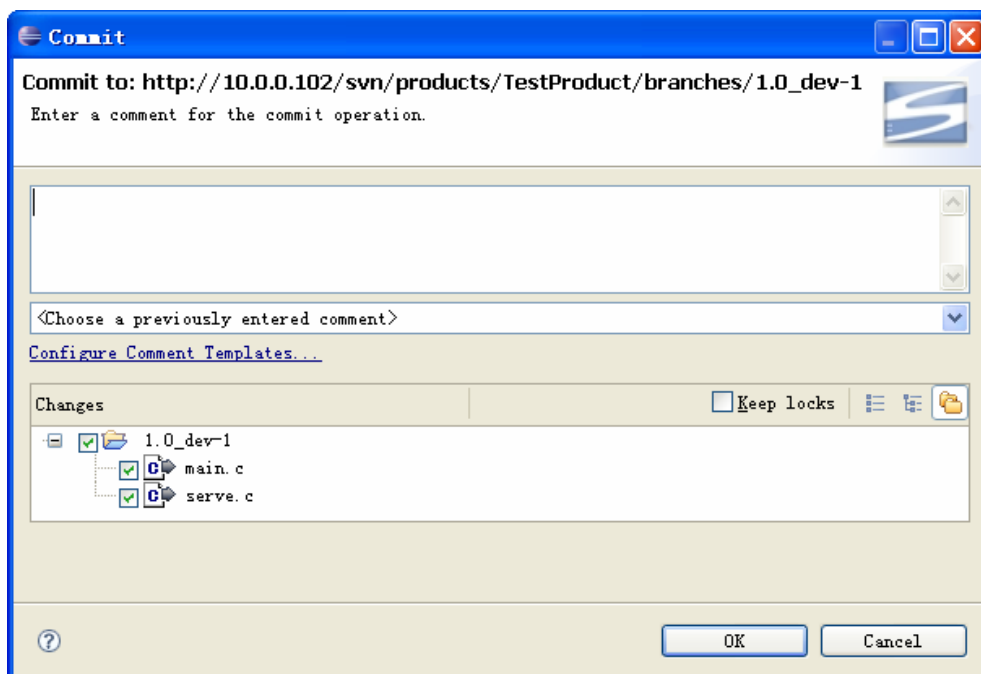
**UI Path: SCMClient > Get Local Copy; SCMClient > Commit...**

To change files on a specific branch, you need to check out files to a local copy, modify the files and then commit changes to the repository branch.



As shown above, select a branch to check out from “Get Branch” drop list.

Once you have made the changes to all the files required, you commit your changes to the repository branch as show below:



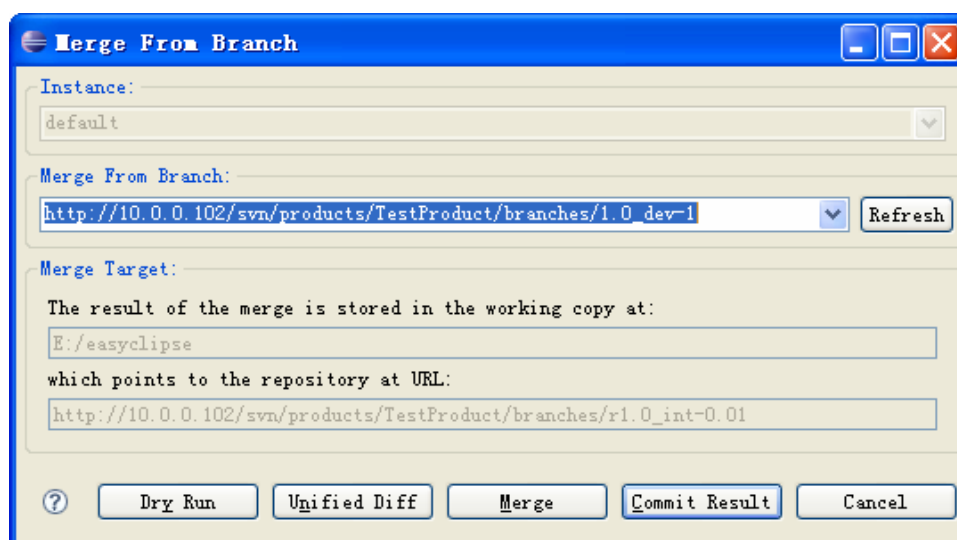
## 2.6. Merge and Conflict Resolution

**UI Path: SCMClient > Merge from Branch; SCMClient > Edit Conflicts; SCMClient > Mark Resolved**

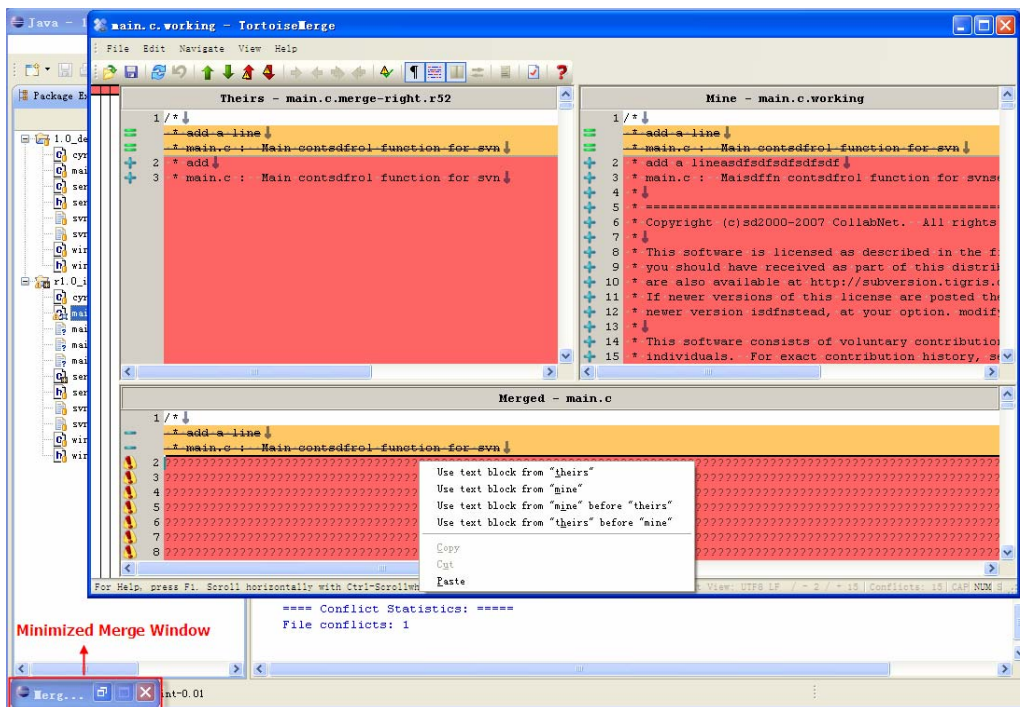
When the changes on your branch is ready to be incorporated to the target branch, you select (highlight) the local copy of the merge-to (target) branch, then go to **SCMClient > Merge from Branch**. In the Merge from Branch window, select an appropriate merge-from branch from the dropdown list. The Merge Target related information is populated automatically based on the local copy branch you have selected earlier. Next, click on the Merge button.

If the merge is successful, then you proceed to commit the local copy of target branch to the repository by clicking on the Commit Result button.

The screenshots below show merging from development branch 1.0\_dev-1 to integration branch r1.0\_int-0.01.



Conflict occurs if two branches changed the same line of the same file, which was merged into the same integration branch. In this case, the latter merge needs to resolve the conflict. For example, assuming branches 1.0\_dev-1 and 1.0\_dev-2 changed the same line of a file, 1.0\_dev-2 completes its merge first without conflict, but when branch 1.0\_dev-1 starts its merge, the merge will fail due to conflict. The owner of branch 1.0\_dev-1 must resolve the conflict on the local copy, mark the file as resolved and then commit. He can temporarily minimize the merge window and start conflict resolution (select **SCMClient > Edit Conflicts**) as shown below:

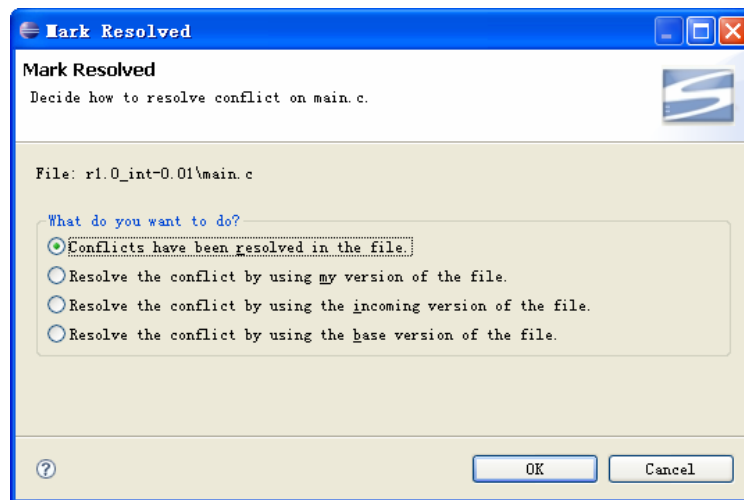


By default (**SCMClient > External Conflict Resolver** is turned on), there are three panes in the Merge Conflict Resolution window. Two are side-by-side on the upper half, and the third one is below the two. The upper-left pane displays the diff'ed file between the one in the common base and the one in the merge-from branch. The upper-right pane displays the diff'ed file between the one in the common base and the one in the merge-to branch.

The third pane displays the file with lines where conflicts occurred. You have two options to modify this file to resolve conflict on this pane:

1. Right click on a line in conflict, and you have four choices as shown in the above screenshot.
2. Edit a line manually and make necessary modifications (You may also do this after completing option 1).

Once all conflicts are resolved in a file, you save the file and mark the conflict is resolved (select **SCMClient > Mark Resolved**) as shown in the screenshot below:



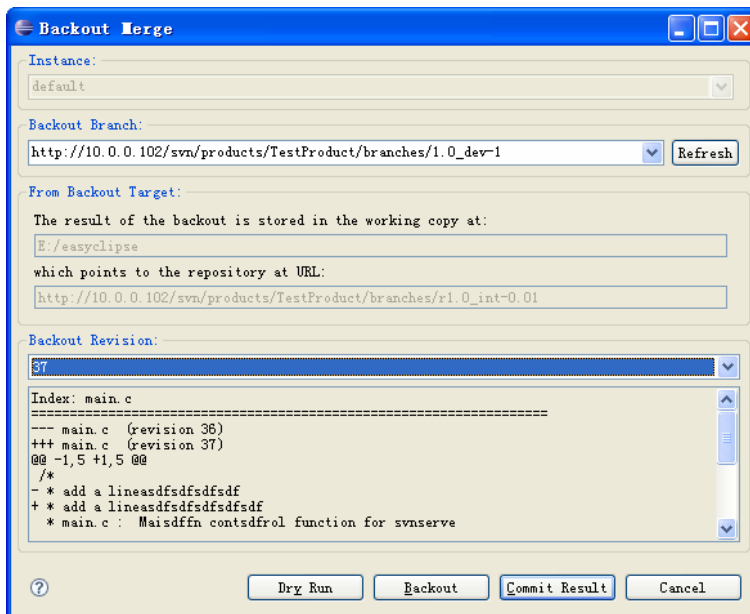
After all files having conflict are resolved and marked, you then restore the minimized Merge Windows and commit merge result by clicking the Commit Result button.

## 2.7. Backout Merge

### UI Path: **SCMClient > Backout Merge**

Easy!Flow SCMClient supports backout of development branch from integration branch. For example, Larry merged branch 1.0\_dev-1 into r1.0\_int-0.01. Later we found out that 1.0\_dev-1 introduced a critical issue that cannot be resolved in a short period of time. In order not to block the integration and release, Larry uses Easy!Flow SCMClient "Backout Merge" feature to back out changes made by branch 1.0\_dev-1 from integration branch r1.0\_int-0.01.

The following example illustrates the backout operation. First, select (highlight) r1.0\_int-0.01 from the Project window (as the local backout target), then select **SCMClient > Backout Merge**. In the Backout Merge window, choose 1.0\_dev-1 as the "Backout Branch". The Backout Target related information is populated automatically based on the local copy branch you have selected in the first step. Next, choose the backout revision number (generated when 1.0\_dev-1 was merged to r1.0\_int-0.01) from "Backout Revision" dropdown list and click on "Backout". Backout will take place in the local copy of r1.0\_int-0.01. The last step is to click on "Commit Result" (to the repository) to complete the operation.

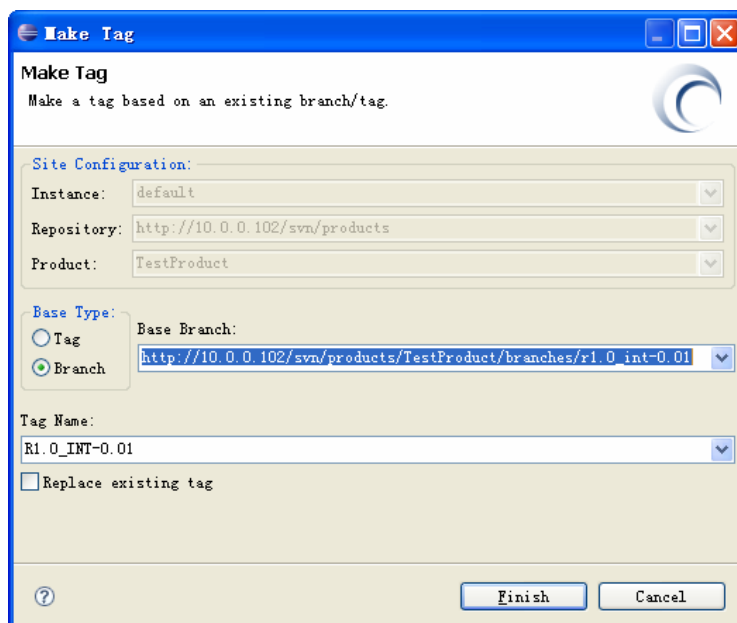


## 2.8. Tagging a Baseline

### UI Path: SCMClient > Make Tag

A Tag, also called label, marks a baseline for a product in the repository. SCMClient supports tagging either on a branch or on another tag. Normally, after integration is completed there should be a tag applied to the integration branch to mark it as a baseline for future work.

**Note:** Only the administrative users can apply official SCM tags. An official SCM tag has the same name as the integration branch but with all the letters in upper case, such as R1.0\_INT-0.01.



### 3. FAQ

- [Can I commit merge/backout result from SCMClient > Commit... rather than from Merge Window?](#)
  - [In Merge/Backout Window, why Merge From Branch droplist lists a file/directory rather than branches?](#)
- 

**Can I commit merge/backout result from SCMClient > Commit... rather than from Merge Window?**

No. Only the "Commit Result" button can call up a Commit Window with a comment in given format, which is required to record the Merge Action in Easy!Flow DB.

**In Merge/Backout Window, why Merge From Branch droplist lists a file/directory rather than branches?**

You did not select/highlight the local copy (Project) of the Merge-to (Target) branch in Eclipse before calling up the Merge Window.