

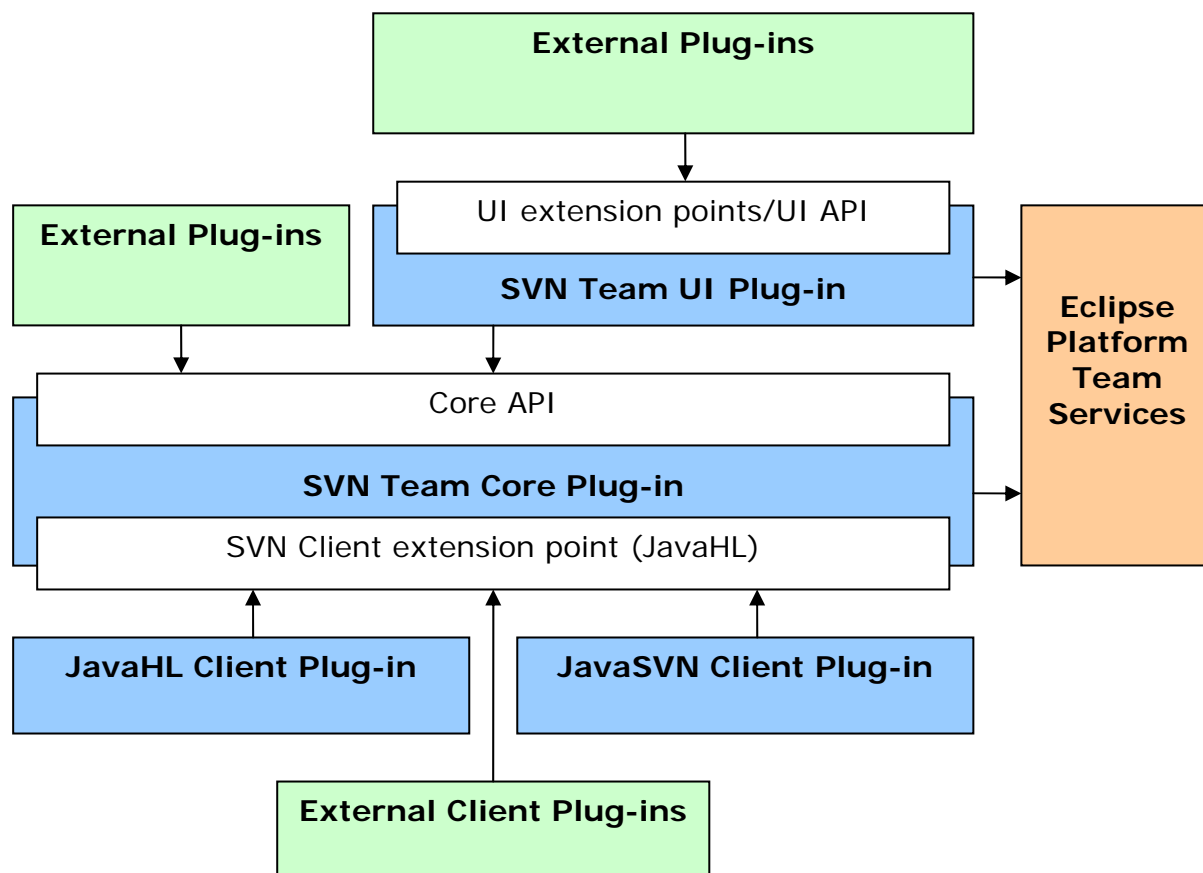
Subversive Architecture Overview

Table of contents

Subversive modules	2
Subversive architecture	3
Sample of Subversive Core API usage	4
The Command Framework functionality	7
Extension points	9
The Subversive Core extension points	9
The Subversive UI extension points	10

Subversive modules

There are two main plug-ins in the Subversive project: Core and UI. Additionally the Subversive distribution contains two SVN Client plug-ins: native – JavaHL and pure Java implementation – JavaSVN. As you can see from the Picture 1 contributors may implement their own SVN Client Library plug-ins and Subversive will use them automatically after its installation into the Eclipse IDE. Also Subversive provides reuse abilities to the external plug-ins designed for automated or interactive work.



Picture 1 Subversive modules diagram

Subversive Core module provides flexible and easy to use API which allows user to interact with all SVN functionality in simple and similar way. At the same time interface simplicity does not make performance impact and user is able to build powerful and high-performance applications on top of the Subversive Core base. Subversive Core is tested in the headless environment and is a solid ground for creation of automated applications.

Subversive UI module is stable and usable. Most significant benefits are:

- Usability optimizations (pop-up menu enablement's, controls layout etc.)
- Detailed description on each UI form
- “On the fly” data validation in dialogs and wizards
- Eclipse IDE-like style
- Extensibility

Last benefit allows users create their own UI extensions for the Subversive. Such extensions can be tracker integrations or any other application that require SVN client base. Additionally the Subversive UI has several extension points which allows contribute into:

- Synchronize View actions
- Checkout action
- Share Project action
- Commit action
- Resources decoration
- History View multi-line comment

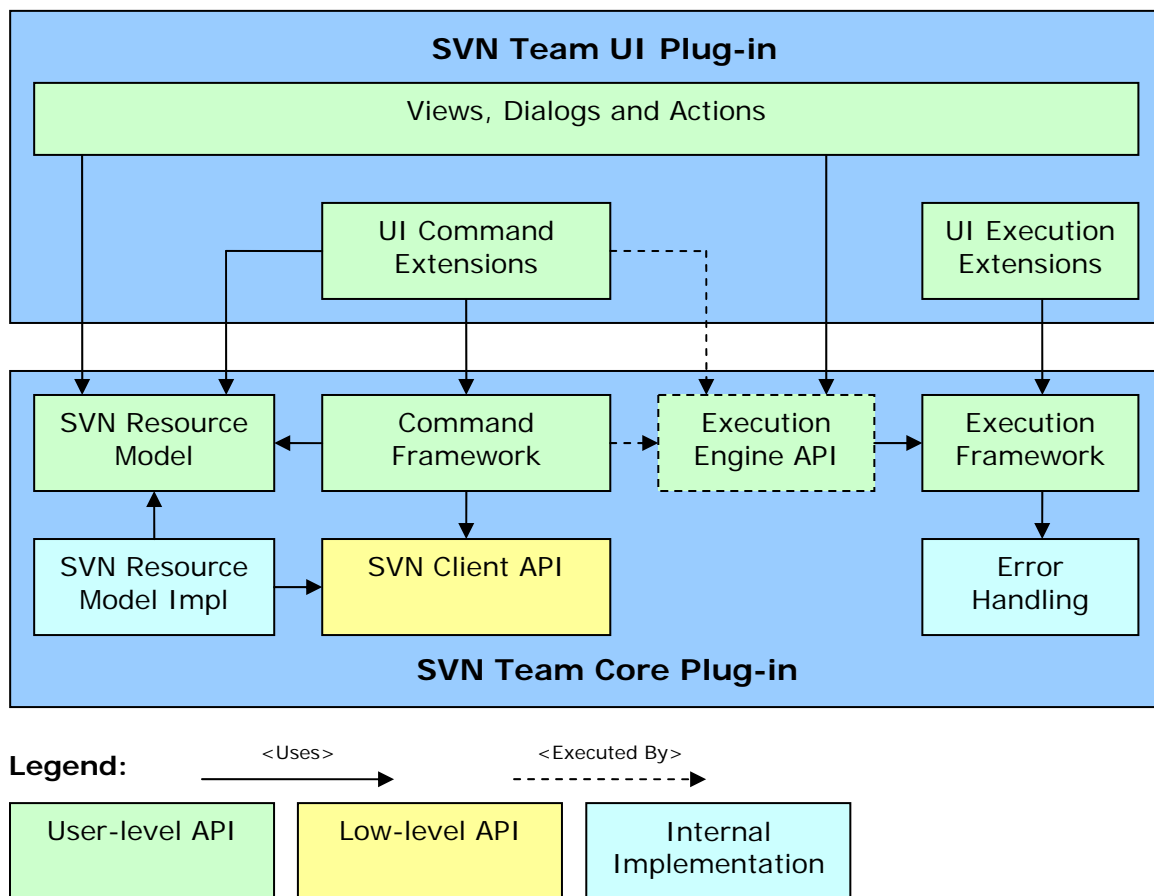
- Error reporting

This list can be extended. Comments and suggestions from community regarding enhancement of the Subversive integration abilities, creation new extension points and API's, are welcome.

Subversive architecture

The Subversive project architecture follows to several important requirements for both – UI and Core modules:

- Precise separating of UI and Core parts
- Unified error handling
- Failure tolerance
- Conceptual integrity of API
- Strong API levels delimitation
- Easy API extensibility



Picture 2 Subversive architecture diagram

Core plug-in has two API levels – user-level and low-level. First is most frequently used API level and it is based on the low-level API.

User-level API contains following parts:

- **Command Framework** designed correspondingly to “Command Pattern” concept. It allows user to perform any complex interactions with Subversion and Eclipse Platform in performance-optimal and easy way. Command Framework already contains command implementations for all frequently used cases of interaction with Subversion, checked out projects and Eclipse Platform. The Command Framework allows reducing “copy-paste”-technique usage and providing fast development with minimal efforts. All provided commands can be fully reused in external tools without any limitations
- **SVN Resource Model** allows building of local and repository resource hierarchies and provide command framework with all required information in one standard way

- **Execution Framework** allows running all commands in the similar way. Its background implementation is responsible for automated resource locking rules calculation and error handling
- **Execution Engine API** is set of classes and interfaces that hides from user how background implementation serves a Commands execution
- **Error Handling** mechanism provided by Subversive Core allows user to build applications with high failure tolerance: one failed command does not prevent other commands from execution if it is required. Moreover, commands itself can be recovered from errors, for example: Commit Command commits resources to all repositories that are available and skip all resources that cannot be committed; all information about committed and uncommitted resources is provided to the caller level.

Low-level API allows user to build Command Framework extensions in order to handle some rarely used or application-specific cases.

The Subversive UI extends Command and Execution Frameworks with UI specific features most of which can be reused by depended projects. Additionally Subversive UI provides powerful and flexible Data Validation Framework for dialogs and wizards. Provided extension points allow reorganizing the Subversive UI functionality in some critical cases. For example “Error Reporting” extension point allows redirect bug reporting into application specific mailing list.

UI plug-in extends Core functionality with several UI-specific features:

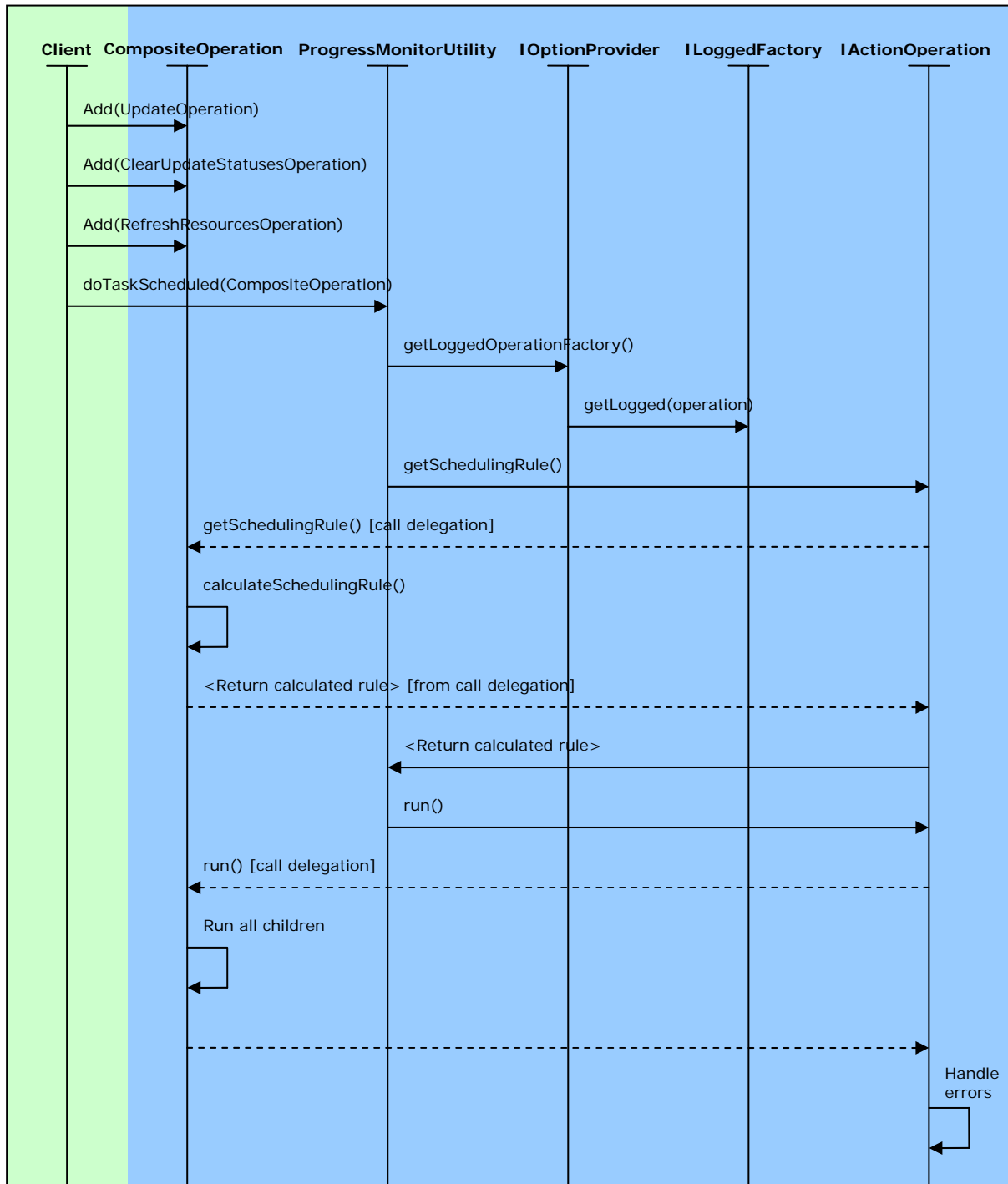
- **UI Execution Extensions:** enhance error handling in order to distinguish errors by severity, show errors to user and propose sending of bug reports to plug-in developers, connect progress monitoring to Eclipse Platform UI
- **UI Command Extensions** include commands that required interaction with Eclipse Platform UI.

The Subversive architecture overview shows how the project structure corresponds to requirements. First of all both modules – Core and UI – are strongly separated and Core module is fully functional and allows user to build automated applications. Unified error handling mechanisms provided by Execution Framework allows improving of the Subversive project failure tolerance. API concept allows easy extending without mixing of different API levels in the same code.

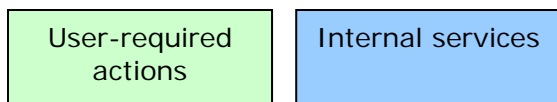
Sample of Subversive Core API usage

On the Picture 3 you can see flow of calls that is required from user in order to update resources to latest revision in background execution thread. And next – Code Sample 1, Code Sample 2 – are samples how it looks in the code.

Background Update Operation Flow



Legend:



Picture 3 Update flow sequence diagram

Code Sample 1 Subversive UpdateAction class implementation

```
public class UpdateAction extends AbstractRecursiveTeamAction {

    public UpdateAction() {
        super();
    }

    public void run(IAction action) {
        IResource []resources = UnacceptableOperationNotificator.
            shrinkResourcesWithNotOnRespositoryParents(
                this.getShell(), this.getSelectedResources(IStateFilter.SF_ONREPOSITORY));
        if (resources == null || resources.length == 0) {
            return;
        }

        this.runScheduled(UpdateAction.getUpdateOperation(this.getShell(), resources));
    }

    protected boolean isEnabled() throws TeamException {
        return this.getSelectedResources(IStateFilter.SF_ONREPOSITORY).length > 0;
    }

    public static CompositeOperation getUpdateOperation(Shell shell, IResource []updateSet) {
        final DetectDeletedProjectsOperation detectOp = new DetectDeletedProjectsOperation(updateSet);
        final UpdateOperation mainOp = new UpdateOperation(detectOp, true);

        IResourceProvider refreshProvider = new IResourceProvider() {
            public IResource []getResources() {
                HashSet fullSet = new HashSet(Arrays.asList(mainOp.getResources()));
                fullSet.addAll(Arrays.asList(detectOp.getDeleted()));
                return (IResource [])fullSet.toArray(new IResource[fullSet.size()]);
            }
        };

        CompositeOperation op = new CompositeOperation(mainOp.getOperationName());

        op.add(detectOp);
        SaveProjectMetaOperation saveOp = new SaveProjectMetaOperation(detectOp);
        op.add(saveOp);
        op.add(mainOp);
        op.add(new RestoreProjectMetaOperation(saveOp));
        op.add(new ProcessDeletedProjectsOperation(detectOp));
        op.add(new ClearUpdateStatusesOperation(refreshProvider));
        op.add(new RefreshResourcesOperation(refreshProvider));
        op.add(new NotifyUnresolvedConflictOperation(shell, mainOp));

        return op;
    }
}
```

As you can see the UpdateAction class implementation is more complex in compare with the sequence diagram because it supports more functionality — detecting projects, that is deleted on repository, saving Eclipse IDE meta-information in order to prevent problems when something like “.project” is deleted on repository.

In general case it is not required for programmer to implement his own commands and work with SVN Client Library API. Nevertheless programmer can create own commands using SVN Client Library API – it also easy. The command implementation does not requires from programmer any additional actions (like integral resource locking policies calculation for all commands, interfaces that allows data transmitting between commands, error handling and crash recovery support) except of freeing of allocated resources in finally section.

Code Sample 2 Command implementation

```
public class ExportOperation extends AbstractRepositoryOperation {
    protected String path;

    public ExportOperation(IRepositoryResource resource, String path) {
        super("Export", new IRepositoryResource[] {resource});
        this.path = path;
    }

    protected void runImpl(IProgressMonitor monitor) throws Exception {
        IRepositoryResource resource = this.operableData()[0];
        IRepositoryLocation location = resource.getRepositoryLocation();
        ISVNClientWrapper proxy = location.acquireSVNProxy();
        try {
            String path = this.path + "/" + resource.getName();
            proxy.doExport(SVNUtility.encodeURL(resource.getUrl()),
                path,
                resource.getSelectedRevision(),
                resource.getPegRevision(),
                true, // force
                false, // ignore externals
                true, // recurse
                null, // native EOL
                new SVNProgressMonitor(this, monitor, null));
        } finally {
            location.releaseSVNProxy(proxy);
        }
    }

    protected String getShortErrorMessage(Throwable t) {
        return "Export operation for '" + this.operableData()[0].getUrl() + "' failed.";
    }
}
```

The Command Framework functionality

The Command Framework totally contains 89 commands which are presents in three subsets:

- Execution Framework Part (2)
- Core Command Framework (60)
- UI Command Extensions (17)

The Core Commands cover all SVN functionality used in Subversive and it can be fully reused without any restrictions. Most UI Commands are designed for interactive cases. So, they cannot be used in automated processing. Execution Framework Commands, like LoggedOperation and CompositeOperation, are responsible for error handling and resource locking rules calculation.

Command	Description
Execution Framework Part (2)	
LoggedOperation	Allows safely write errors to log
CompositeOperation	Provides the way to combine different operations
Core Command Framework (60)	
SaveProjectMetaOperation	Saves project meta (.project and .classpath) in order to prevent project refresh problem when meta is deleted
RestoreProjectMetaOperation	Restores project meta (.project and .classpath) in order to prevent project refresh problem when meta is deleted
ShareProjectOperation	Shares the project from scratch
ReconnectProjectOperation	Reconnects the projects with existing SVN meta-information
DisconnectOperation	Disconnects the projects with or without deletion of SVN meta-information
CheckoutOperation	Checkout set of projects into workspace
CheckoutAsOperation	Checkout project into specified location with specified set of options
ObtainProjectNameOperation	Request real project name for the project in SVN repository
CommitOperation	Commit resources
JavaHLMergeOperation	Merge resources in standard way
MergeOperation (experimental)	Interactive merge implementation
MergeStatusOperation (experimental)	Interactive merge implementation
UpdateOperation	Update resources
AddToSVNIgnoreOperation	Add resources to svn:ignore
AddToSVNOperation	Add resources to source control
LockOperation	Lock resources
UnlockOperation	Unlock resources
RevertOperation	Revert modifications
MarkAsMergedOperation	Mark conflicts as resolved
RemoveNonVersionedResourcesOperation	Remove any non-versioned resources starting from the level specified
SwitchOperation	Switch project to new URL
GetPropertiesOperation	Get all resource properties
SetPropertyOperation	Set resource property
RemovePropertyOperation	Remove resource property
GetAllResourcesOperation	Get all resources for the specified local folder including deleted, missing etc.
DetectDeletedProjectsOperation	Detect which projects are deleted on repository (the deleted projects cannot be processed in normal way)
SaveRepositoryLocationsOperation	Save Subversive meta-information changes
DiscardRepositoryLocationsOperation	Remove specified repository locations from the Subversive meta-information
AddRepositoryLocationOperation	Add repository location to the Subversive meta-information
AddRevisionLinkOperation	Create revision links in the Subversive meta-information
RemoteStatusOperation	Update status for the specified resources
InfoOperation	Retrieve Info2 structure for the specified resource
RelocateWorkingCopyOperation	Relocate working copy

CreatePatchOperation	Create patch based on working copy changes
RefreshResourcesOperation	Refresh workspace tree and send internal Subversive resource modification events
NotifyProjectStatesChangedOperation	Send internal Subversive notification when project state is changed (shared, disconnected, opened, closed etc.)
GetRemoteContentsOperation	Get remote file or folder contents into specified folder overriding existing files
GetFileContentOperation	Fetch remote file content from SVN
GetLocalFileContentOperation	Fetch local file content from SVN (BASE or WORKING revisions)
CleanupOperation	Cleanup working copy after power loss or other failure
ClearLocalStatusesOperation	Refresh status cache for the specified resources
MoveResourceOperation	Move resources between folders in one/different working copy/copies saving the history
CopyResourceWithHistoryOperation	Copy resources between folders in one/different working copy/copies saving the history
CopyResourceOperation	Copy resources without saving history
DeleteResourceOperation	Delete versioned resources
RenameResourceOperation	Move resource from one URL to another
LocateProjectsOperation	Find Eclipse projects on repository
ImportOperation	Import specified folder into repository
GetResourceAnnotationOperation	Get annotation for the specified resource
GetRemotePropertiesOperation	Get properties for the resource on repository
GetLogMessagesOperation	Get resource modification history
ExportOperation	Export repository resource into specified local folder
DeleteResourcesOperation	Delete resources directly from repository
CreatePatchOperation (remote)	Create patch bases on difference between revisions
CreateFolderOperation	Create set of folders at any depth on the repository
CreateFileOperation	Create file directly on the repository with specified initial content
BreakLockOperation	Unlock resource directly on the repository
BranchTagOperation	Create branch or tag
CopyResourcesOperation (remote)	Copy resources to specified URL
MoveResourcesOperation (remote)	Move resources to specified URL
UI Command Extensions (17)	
UILoggedOperation	UI extension of LoggedOperation, show errors to user and propose to send bug report in case of internal failures
ShowUpdateViewOperation	Show synchronize view
ShowConflictEditorOperation	Show conflicted files editor (for resources update by external tools)
ProcessDeletedProjectsOperation	Notify user about deleted projects and request his decision
ClearUpdateStatusesOperation	Clear update statuses cached in Synchronize View
ClearMergeStatusesOperation (experimental)	Clear merge statuses cached in interactive Merge View
ShowPropertiesOperation	Show property editor
RefreshRepositoryLocationsOperation	Refresh repository browsing view
PrepareRemoteResourcesTransferrableOperation	Insert references to repository resources into clipboard
PasteRemoteResourcesOperation	Paste repository resources from clipboard into selected location
OpenRemoteFileOperation	Open remote file in its default viewer
NotifyUnresolvedConflictOperation	Notify user about unresolved conflicts in time of updating/committing resources
ShowMergeViewOperation (experimental)	Show interactive Merge View
FileToClipboardOperation	Copy file content into clipboard
CompareResourcesOperation	Three-way compare of working copy resources with the selected revision and show the result in compare viewer
CompareRepositoryResourcesOperation	Two-way compare of the repository resources with specified revisions and show the result in compare viewer
RefreshRemoteResourcesOperation	Refresh repository resources in the repository browsing view

Extension points

The Subversive project provides several extension points:

- SVN Client Library
- Mail Settings Provider
- Synchronize View Actions Contribution
- Share Project Wizard
- Multi-line Comments in History
- Checkout
- Commit

An interface of the first two extension points is full-featured and enough flexible from our point of view. It covers most possible integration aspects and can be treated as stable. Please note that we plan to move “Mail Settings Provider” to Core plug-in in order to allow mail reporting for the automated applications.

Extensions are subjects of further discussions and we will very appreciate to community for any ideas on how to improve them.

The Subversive Core extension points

Core plug-in provides extension point that allows contributors to implement alternative SVN Client Library support. The extension should implement following interface:

Interface 1 “SVN Client Library” extension point

```
public interface ISVNClientWrapperFactory {
    public static final String DEFAULT_ID = "org.polarion.team.client.javahl.core";

    /**
     * Makes new SVN Client Library instance
     * @return SVN Client Library instance
     */
    public ISVNClientWrapper newInstance();

    /**
     * Returns unique SVN Client library plug-in id
     * @return SVN Client library plug-in id
     */
    public String getId();

    /**
     * Returns user-friendly SVN Client library plug-in name
     * @return SVN Client library plug-in name
     */
    public String getName();

    /**
     * Tell revision change reporting for folders is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isReportRevisionChangeAllowed();

    /**
     * Tell compare folders is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isCompareFoldersAllowed();

    /**
     * Tell interactive merge is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isInteractiveMergeAllowed();

    /**
     * Tell atomic commit is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isAtomicCommitAllowed();

    /**
     * Tell fetch locks is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isFetchLocksAllowed();

    /**
     * Tell SSH settings is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isSSHOptionsAllowed();

    /**
     * Tell proxy settings is allowed or not
     * @return true if allowed false otherwise
     */
    public boolean isProxyOptionsAllowed();
}
```

ISVNClientWrapper interface, which instances is returned by ISVNClientWrapperFactory .newInstance() method, is constructed very similar to the JavaHL interface and allows to hide specific of the SVN Client Library interface from the Subversive Core module.

The Subversive project uses some specific features provided by JavaSVN library, at the same time the features are unsupported by current implementation of the native JavaHL library. In general case an arbitrary SVN Client Library plug-in may provide partial support of extended features. So, we have the compatibility problem with the Subversive-specific features. Subversive architecture allows solving the problem in simple way. All compatibility settings are provided by each SVN Client Library plug-in through ISVNClientWrapperFactory interface.

If a SVN Client Library plug-in does not support extended features it reflects on the Subversive functionality like described below:

- The “Cross WC atomic commit” feature implementation is completely transparent for end users who are used the Subversive and for programmers who are used the Subversive API. One little difference is non-atomic revision numbers in case when feature is inaccessible
- Compare folders is inaccessible
- SSH and Proxy settings is inaccessible
- Locks decoration for repository resources is inaccessible
- Interactive merge is inaccessible
- “Report revision change for folders” option is inaccessible.

The Subversive UI extension points

The Subversive UI plug-in provides a set of different extension points:

- **“Mail Settings Provider”** extension point allows customizing and redirecting of “Automated Bug-Reporter” Subversive project service.

Interface 2 “Mail Settings Provider” extension point

```
public interface IMailSettingsProvider {
    /**
     * Returns report addressee
     * @return report addressee
     */
    public String getEmailTo();
    /**
     * Returns report sender
     * @return report sender
     */
    public String getEmailFrom();
    /**
     * Returns plug-in name
     * @return plug-in name
     */
    public String getPluginName();
    /**
     * Returns plug-in version
     * @return plug-in version
     */
    public String getProductVersion();
    /**
     * Returns mail server host
     * @return mail server host
     */
    public String getHost();
    /**
     * Returns mail server port
     * @return mail server port
     */
    public String getPort();
}
```

- **“Synchronize View Action Contributions”** extension point allows adding of custom actions into the Subversive project Synchronize View

Interface 3 “Synchronize View Action Contributions” extension point

```
public interface ISynchronizeViewActionContributor {
    /**
     * This method returns synchronize view action contributions for update mode
     * @return collection of AbstractSynchronizeActionGroup
     */
    public Collection getUpdateContributions();
    /**
     * This method returns synchronize view action contributions for merge mode
     * @return collection of AbstractSynchronizeActionGroup
     */
    public Collection getMergeContributions();
}
```

- **“Share Project Wizard”** extension point allows overriding of the default Subversive project behavior while sharing the project.

Interface 4 “Share Project Wizard” extension point

```
public interface IShareProjectFactory {
    /**
     * The method provides ShareProjectWizard page with some extended options in compare to default Subversive implementation
     * @param project the project which will be shared
     * @return wizard page
     */
    public SelectProjectNamePage getProjectLayoutPage(IProject project);
    /**
     * Allows to override default Subversive behavior while sharing the project
     * @param project will be shared
     * @param location the repository location which will be used in order to share the project
     * @param page advanced share project configuration page
     * @return share project operation implementation which overrides default Subversive behavior
     */
    public ShareProjectOperation getShareProjectOperation(IProject project, IRepositoryLocation location, SelectProjectNamePage
page);

    /**
     * Force disablement of the finish button on the "Already Connected" page
     * @return true if should be disallowed
     */
    public boolean disallowFinishOnAlreadyConnected();
    /**
     * Force disablement of the finish button on the "Add Repository Location" page
     * @return true if should be disallowed
     */
    public boolean disallowFinishOnAddRepositoryLocation();
    /**
     * Force disablement of the finish button on the "Select Repository Location" page
     * @return true if should be disallowed
     */
    public boolean disallowFinishOnSelectRepositoryLocation();
}
```

- “Multi-line Comments in History” extension point allows replacement of the default Subversive multi-line viewer implementation to more powerful which, for example, provides hyperlinks in comments etc.

Interface 5 “Multi-line Comments in History” extension point

```
public interface IHistoryViewFactory {
    /**
     * Returns project-specific multi-line comment view implementation
     * @return project-specific multi-line comment view implementation
     */
    public ICommentView getCommentView();
}
```

- “Checkout” extension point allows performing of some non-standard actions with projects which will be checked out by the product that contributes the Subversive project.

Interface 6 “Multi-line Comments in History” extension point

```
public interface ICheckoutFactory {
    /**
     * The method allows specific decorations for the projects in Checkout As wizard
     * @param name2resources mapping between proposed project names and repository resources that is referenced
     * to corresponding projects on repository
     * @return table decorator
     */
    public ITableLabelProvider getLabelProvider(Map name2resources);
    /**
     * The method provides specific filter allowing automated detection of the projects on repository
     * @return repository resource filter
     */
    public LocateProjectsOperation.ILocateFilter getLocateFilter();
    /**
     * The method allows override the default Subversive project Checkout Operation behavior with specific one
     * @param shell the Shell instance that will be used to interact with user
     * @param remote resources that will be checked out
     * @param checkoutMap project names mapping
     * @param respectHierarchy create locally folder structure that corresponds to repository projects layout
     * @param location destination folder
     * @param checkoutRecursively true if recursive checkout is required, false otherwise
     * @return alternative Checkout Operation instance
     */
    public IActionOperation getCheckoutOperation(Shell shell, IRepositoryResource []remote,
Map checkoutMap, boolean respectHierarchy, String location, boolean checkoutRecursively);
    /**
     * The method allows correction of the automatically proposed project name mapping
     * @param name2resources automatically proposed project name mapping
     * @return corrected project name mapping
     */
    public Map prepareName2resources(Map name2resources);
    /**
     * The method allows providing of some additional processing for the projects found on repository
     * @param op default locate projects operation
     * @param provider found repository resource provider
     * @return additional resources provider
     */
    public IRepositoryResourceProvider additionalProcessing(CompositeOperation op, IRepositoryResourceProvider provider);
}
```

- “Commit” extension point allows overriding the standard Subversive Commit Dialog with more powerful and performing additional tasks for the committed resources.

Interface 7 “Commit” extension point

```
public interface ICommitActionFactory {  
  
    /**  
     * The method provide abilities in extending of the standard Subversive Commit Dialog to more powerful  
     * @param shell Shell instance which will be used to interact with user  
     * @param allFilesToCommit full set of files which will be committed  
     * @param panel the default Subversive Commit Panel implementation  
     * @return enhanced Commit Dialog  
     */  
    public ICommitDialog getCommitDialog(Shell shell, Collection allFilesToCommit, ICommentDialogPanel panel);  
  
    /**  
     * The method allows customizing of the Commit Operation  
     * @param operation prepared Commit operation  
     * @param revisionProvider committed revision provider  
     * @param dependsOn dependencies which can prevent commit operation execution in case of failure  
     * @param part workbench part which will be used to interact with user  
     */  
    public void performAfterCommitTasks(CompositeOperation operation, IRevisionProvider revisionProvider,  
        IActionOperation[] dependsOn, IWorkbenchPart part);  
}
```